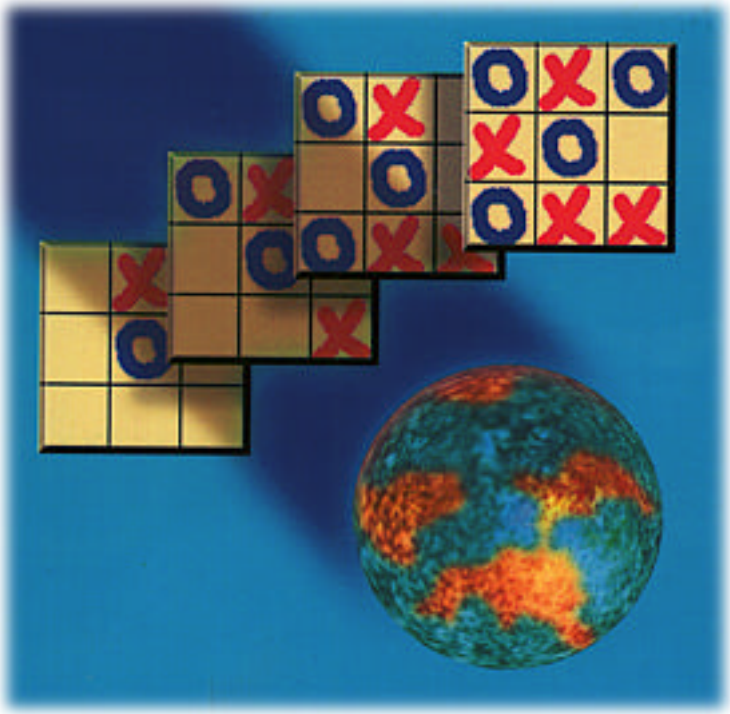


PRO-
KURSSI

Petri Kiuttu

JAVA
-OHJELMOINTI

Suomen Atk-*kustannus*

Java-ohjelmointi – Pro-kurssi

Kaikki oikeudet pidätetään. Tämän julkaisun jäljentäminen ilman tekijän kirjallista lupaa painamalla, monistamalla, äänittämällä tai muulla tavoin on tekijänoikeuslain mukaisesti kielletty.

Copyright

Kirjoittajat	Petri Kiuttu
Kustantaja	Suomen Atk-kustannus Oy PL 68 02601 ESPOO Puhelin: 09-4765 8550 Telekopio: 09-5765 8595 Sähköposti: palvelu@satku.fi Internet: www.satku.fi
Kansi	Frank Chaumont / Timo Ravela
Taitto	Tekijä
ISBN	951-762-548-0
Sähköinen laitos	1. Sähköinen laitos 1998 Helsinki Painettu laitos Jyväskylä 1997

Sisällysluettelo

1. JAVA-MAAILMA 15

1.1 Mitä ovat Java ja JavaScript? 15
1.2 Javaa tukevat selaimet 16
1.3 Hotjava 17
1.4 Javan ja JavaScriptin mahdollisuudet 19
1.5 Turvallisuus 21
1.6 Kehitysympäristöt 21
1.7 Sovelluskehittimet 23

2. JOHDATUS JAVA-KIELEEN 25

2.1 Ominaisuudet 25
2.2 Ohjelmointi 26
2.3 Kirjoita kerran, aja missä vain 26
2.4 JavaScript-kieli 27
2.5 Java-sovellukset 27
2.6 Ensimmäinen Java-sovellus 28
2.7 Lähdekoodin kääntäminen JDK:lla 29
2.8 Editori 30

3. JAVA-KIELEN PERUSTEET 33

3.1 Java- ja C++-kielen eroja 33
3.2 Java-kielen kirjoitusasu 34
3.3 Kommentit 35
3.4 Muuttujat ja tietotyypit 36
3.5 Tyyppimuunnokset 39
3.6 Lukujärjestelmät 41
3.7 Merkkijonot 42
3.8 Merkkijonojen käsittely 43
3.9 Taulukot ja niiden käsittely 46
3.10 Operaattorit 48
3.11 Ohjauslauseet 51
3.12 Olio-ohjelmointi 58

3.13 Luokka ... 60
3.14 Luokan muuttujat ... 62
3.15 Luokan funktiot ... 64
3.16 Luokan käyttöönotto ... 65
3.17 Muodostimet ... 67
3.18 Periytyvyys ... 70
3.19 Tyyppimuunnokset olioilla ... 72
3.20 Ylikuormittaminen ... 73
3.21 Ylikirjoittaminen ... 74
3.22 This-muuttuja ... 74
3.23 Super-muuttuja ... 75
3.24 Staattiset muuttujat ja funktiot ... 76
3.25 Rajapinnat ... 77
3.26 Luokkakirjastot ... 78
3.27 Luokkakirjasto java.lang ... 80
3.28 Poikkeustilanteiden käsittely ... 82

4. SOVELLUSOHJELMOINTI ... 85
4.1 Tietovirrat ... 85
4.2 Muita System-luokan funktioita ... 87
4.3 I/O-toiminnot ... 88
4.4 Ohjelmien suorittaminen ... 91
4.5 Ikkunointia ... 92
4.6 Liityntä käyttöjärjestelmään ... 94
4.7 Useasta luokasta koostuva sovellus ... 94
4.8 Sovellukset appletteina ... 95

5. APPLETIN RAKENNE ... 99
5.1 Appletit ... 99
5.2 Appletin käyttöönotto ... 100
5.3 JAR-tiedostot ... 102
5.4 Applet-luokka - ohjelmien perusluokka ... 103
5.5 Tapahtumafunktioista ... 105
5.6 Appletin testaaminen ... 106
5.7 Parametrien lukeminen ... 109

5.8 Appletin URL-osoitteet 111
5.9 Appletit sovelluksina 113

6. INTERAKTIIVISUUS 119
6.1 JDK-versioiden eroja tapahtumankäsittelyssä 119
6.2 JDK1.0 - Tapahtumaluokka Event 120
6.3 JDK1.1- tapahtumien delegointi 126
6.4 Tapahtumaluokat 126
6.5 Tapahtumien sallinta 127
6.6 Kuuntelijat 128
6.7 Komponentit kuuntelijoina 130
6.8 Adapterit 131
6.9 Tapahtumien prosessointi 133
6.10 Interaktiivisuutta elementeillä 134
6.11 Näppäimistön luku 136
6.12 Hiiren seuranta 139

7. GUI:N PERUSTEET 145
7.1 Paneelit 145
7.2 Container 149
7.3 Scrollpane 150
7.4 Komponentit 153
7.5 Layoutit 157
7.6 BorderLayout 159
7.7 CardLayout 160
7.8 FlowLayout 163
7.9 GridLayout 164
7.10 GridBagLayout 166
7.11 Layouttien yhdistäminen 170

8. IKKUNAT, VALIKOT JA DIALOGIT 173
8.1 Ikkunat 173
8.2 Ikkunoiden toiminnot 179
8.3 Valikot 181
8.4 Valikot ja näppäimistö 184

8.5 Popup-valikko 186
8.6 Valikon muokkaaminen 188
8.7 Valikkokomennon suorittaminen 191
8.8 Dialogit 195
8.9 Tiedostodialogit 199
8.10 Tulostaminen 203

9. GRAAFISET ELEMENTIT 207
9.1 Painonappi 207
9.2 Valintalaatikko 210
9.3 Radionappi 213
9.4 Tekstikenttä 216
9.5 Tekstilaatikko 219
9.6 Vierityspalkki 224
9.7 Nimiö 228
9.8 Valintalista 231
9.9 Lista 234
9.10 Canvas-elementti piirtopintana 238
9.11 Elementtien sijoittelu 239
9.12 Lomakkeet 241

10. SÄIKEET JAVASSA 247
10.1 Mitä ovat säikeet? 247
10.2 Miksi säikeitä tarvitaan? 247
10.3 Säie-ohjelmointi Thread-luokalla 248
10.4 Runnable-rajapinta 250
10.5 Säikeen tilan ohjaus 252
10.6 Synkroniset säikeet 255

11. GRAFIIKKA 259
11.1 Piirtopinta 259
11.2 Piirtopinnalle piirtäminen 259
11.3 Värien asettaminen 261
11.4 Piirtofunktiot 263
11.5 Täyttöfunktiot 269

11.6 Alueiden käsittely 276
11.7 Tekstin tulostaminen 278
11.8 Tekstin tulostaminen eri fonteilla 281
11.9 Tekstidemo 283

12. MULTIMEDIAA JAVALLA 289
12.1 Kuvat 289
12.2 Kuvien muokkaus 291
12.3 Äänet 297
12.4 Animaatio 299
12.5 Värinätön animaatio 303
12.6 Kaksoispuskurointi 307
12.7 Multimediaa 316

13. ESIMERKKIOHJELMIA 321
13.1 Othello 321
13.2 Kello 334
13.3 Scrolleri 337

Johdanto

Tervetuloa Java-ohjelmoinnin pariin! Kädessäsi on nyt Java-ohjelmoinnin PRO-kurssikirja. Tämä kurssikirja on kevyt versio Opeta itsellesi Java-ohjelmointi -kirjan kolmannesta painoksesta. Kirja käsittelee kokonaisuudessaan Java 1.1 -ohjelmointia.

Tämä kurssikirja kertoo Java-ohjelmoinnista kaiken oleellisen, jonka tarvitset siirtyessäsi Java-ohjelmoinnin pariin. Et tarvitse ennestään kokemusta ohjelmoinnista, etkä tuntemusta käyttöjärjestelmistä. Riittää, kun olet tutustunut WWW:hen, esimerkiksi Netscapella. Hyödyksi on tietenkin se, että tunnet jonkin ohjelmointikielen (esim. basic) ja osaat tehdä WWW-sivuja HTML *(Hypertext Markup Language*) -kielellä.

Kirjan kohderyhmä

Tämä opas on tarkoitettu juuri sinulle, joka olet kiinnostunut Internetistä ja omien kotisivujen ylläpidosta ja kehittämisestä. Tai sinulle, joka olet kiinnostunut sovellusohjelmoinnista. Java-kielen avulla opit nopeasti tekemään näyttäviä interaktiivisia ja multimediaesityksiä WWW-sivuille.

Kirjasta on ensisijaisesti hyötyä

- niille, jotka osaavat tehdä HTML-kielellä dokumentteja ja haluavat tutustua WWW:n syvällisempään ohjelmointiin
- niille, jotka ymmärtävät CGI-ohjelmointia ja haluavat luoda välitöntä interaktiivisuutta
- niille, jotka ovat opiskelleet jotakin ohjelmointikieltä, esimerkiksi C-kieltä ja haluavat opiskella uuden ajan käyttöjärjestelmästä riippumatonta tietoverkkoihin suunnattua ohjelmointikieltä. Javalla voi toteuttaa täysipainoisia sovelluksia.

Ohjelmointivälineet

Tämän kirjan hyödyntämiseksi tarvitset seuraavat ohjelmointivälineet

- Java-ohjelmointipaketti eli JDK
- Editori (käyttöjärjestelmässä vakiona). CD:llä tulee mukana Editori-niminen, Java-kielellä toteutettu, ohjelmointiin tarkoitettu tekstieditori.
- WWW-selain (ei pakollinen) tai AppletViewer (JDK:n mukana)

Ohjelmointivälineistä tärkein on JDK (*Java Developers Kit*) eli Java-ohjelmointipaketti. Sitä ei tule tämän kirjan mukana CD:llä, koska sen levittäminen CD:llä on kiellettyä. JDK haetaan internetistä SUN:in
sivuilta. Osoite on:

java.sun.com/download/

- versioksi valitaan 1.1

JDK on pakattu tiedosto, joka pitää purkaa ennen käyttöä. JDK 1.1 -versiosta lähtien purkaminen on helpompaa, koska purkaminen onnistuu velhon (*wizard*) avulla. Esimerkiksi Windowsille JDK asennetaan siten, että ajetaan internetistä haettu tiedosto ja seurataan velhon ohjeita. Tiedoston ajaminen tapahtuu valitsemalla esimerkiksi Start-valikosta Run-komennolla.

Purkamisen jälkeen kiintolevyllä on juuressa alihakemisto \java1.1.2 tai vastaava. Samaiseen alihakemistoon kannattaa purkaa lisäksi Javan API (*Application Programming Interface*), joka on joukko HTML-tiedostoja). API ei ole itsepurkautuva, joten purkaminen pitää tehdä esimerkiksi WinZip-ohjelmalla. Api:n saat haettua samasta internet-osoitteesta. API:n purkaminen tuottaa java1.1.2-hakemistoon docs-alihakemiston.

bin		File Folder	24.6.1996 17:39
demo		File Folder	24.6.1996 17:39
docs		File Folder	24.6.1996 17:40
include		File Folder	24.6.1996 17:39
lib		File Folder	24.6.1996 17:39
Changes	83KB	File	28.5.1997 14:30
COPYRIGHT	1KB	File	28.5.1997 14:30
index.html	3KB	HTML File	28.5.1997 14:30
License	6KB	File	28.5.1997 15:13
Readme	35KB	File	28.5.1997 14:30

Toinen ohjelmointiväline on ASCII:ta tallentava tekstieditori. Sellaiseksi käy hyvin käyttöjärjestelmässä mukana tullut editori (Windowsissa Muistio, Unix:ssa vi, pico ja Macissa

Teksturi). Puhtaan tekstieditorin käyttö ei ole niin mukavaa kuin ohjelmointiin tarkoitetun. Sen vuoksi kirjan mukana tulee käyttökelpoinen editori, jolla voi helposti avata, tallentaa lähdekoodeja sekä tietenkin editoida niitä. Sokerina pohjalla on kääntäjän, Appletviewerin ja java-tulkin käyttö suoraan valikoista. Editori osaa myös tuottaa perus-HTML-tiedoston AppletViewerille.

Kolmas, mutta ei pakollinen, on tietenkin Javaa ymmärtävä WWW-selain, esimerkiksi Internet Exploder 4.0 tai Hotjava Beta. Hotjavan voi imuroida edellä kerrotusta osoitteesata http://java.sun.com/download/. Applettien katsomiseen voi käyttää myös ohjelmointipaketin mukana tulevaa Appletviewer-ohjelmaa.

Lukuohje

Kirjassa yleisesti käyttyjä merkintöjä ovat:

Kursivointi	Suomennoksia vastaavat englanninkieliset termit ja huomatukset
Arial	WWW-osoitteet ja valikoista valittavat komennot (eli valikkokomennot). Sekä lisäksi kaikki toimivat www-sivut esimerkkiohjelmille.
`Courier New`	Ohjelma- ja koodiesimerkit
`Courier New ja lihavointi`	Funktiot, jotka esitellään tai funktiot, joita tarvitaan tietyissä toimenpiteissä.

Kirjan rakenne

Kirjaa alkaa yleisellä esittelyllä. Luku 1 perehdyttää lukijan Java-maailmaan, esitellen Javaa tukevia WWW-selaimia, Javan käyttökohteita, ominaisuuksia ja tulossa olevia uudistuksia.

Luku 1	Esittelee Java-maailmaa

Java-kielen esittelyyn on varattu luku 2. Se johdattaa pehmeästi lukijan Java-kielen ominaisuuksiin. Luvun aikana tehdään myös ensimmäinen esimerkkiohjelma.

Luku 2	Esittelee Java-kieltä

Luvussa kolme tutustutaan kielioppiin, jonka ymmärtäminen on perustana Java-ohjelmoinnille. Ei kannata huolestua, jos asiat tuntuvat aluksi liian vaikeilta. Sitä mukaa, kun Java-ohjelmointia oppii enemmän kirjan edetessä (4 -13), voi vilkuilla tästä luvusta kielioppia.

Luku 3	Opastaa Java-kielen kielioppiin

Luvusta 4 lähtien tutustutaan sovellusohjelmointiin. Sovellukset ovat käyttöjärjestelmän päällä toimivia ohjelmia, jotka eivät tarvitse www-selainta.

Luku 4	Tutustuttaa lukijan sovellusohjelmointiin

Viidennessä luvussa kerrotaan Java-appleteista. Java-appletit ovat Java-ohjelmia, jotka tarvitsevat www-selaimen.

Luku 5	Opettaa applettien periaatteet

Lukuihin 6 - 12 kannattaa tutustua siinä järjestyksessä kuin ne ovat kirjassa esitetty. Vaikka näiden lukujen esimerkit ovat toteutettu silmälläpitäen applet-ohjelmointia, kaikki esitetetyt asiat pätevät sovellusohjelmoinnissa.

Luku 6	Käsittelee interaktiivisuutta
Luku 7	Käsittelee ohjelman käyttöliittymän rakentelua
Luku 8	Kertoo ikkunoista, valikoista ja dialogeista
Luku 9	Esittelee graafisia elementtejä
Luku 10	Perehtyy säie-ohjelmointiin
Luku 11	Kertoo grafiikkaominaisuuksien hyödyntämisestä
Luku 12	Asiaa kuvista, äänistä, animaatiosta ja multimediasta

JDK1.1 - Mitä uutta?

Kirja käsittelee Java 1.1-versiota. Mitä uutta, mitä eroja tämä uusi versio tuo tullessaan verrattuna aikaisempaan 1.0-versioon. Se selviää tässä luvussa.

- Kansainvälistyminen (*Internationalization*). 1.1-versio sallii ohjelmoijien kirjoittaa ohjelmat useille eri kielille.

- Sähköisen allekirjoituksen avulla ohjelmista tulee entistä turvallisimpia käyttäjille.
- Java-ohjelmat voivat hyödyntää nyt paremmin käyttöjärjestelmän ominaisuuksia. Esimerkiksi sovellukset voivat tulostaa kirjoittimelle.

- JavaBeans eli Javan komponenttimalli. Sen avulla kokonaisia Java-ohjelmia voidaan rakentaa valmiista komponenteista.

- Kehittyneemmät verkko-ominaisuudet, parempi suorituskyky ja BSD-tyyliset socketit.

- I/O-ominaisuuksia on paranneltu lisäämällä merkkivirrat, jotka vastaavat 16-bittistä UNICODEa.

- Uusi matematiikkapaketti sisältää kaksi luokkaa: BigInteger ja BigDecimal.

- Etäismetodikutsu (*Remote Method Invocation*) eli lyhyesti RMI sallii ohjelmoijien tehdä Java-ohjelmia, jotka sijaitsevat eri palvelimilla ja joiden sisältämiä metodeja voidaan kutsua.

- Olioiden serialisointi (*Object Serialization*) laajentaa Javan I/O-luokkia ymmärtämään verkon yli ladattavia tavuvirtana olevia olioita.

- JDBC-tietokanta liittymä mahdollistaa SQL-yhteensopivien tietokantojen hyödyntämisen Javassa.

- Sisäkkäiset luokat (*Inner Classes*) mahdollistaa helpomman tavan sovitinluokkien (*adapter class*) toteuttamiseen. Sovitinluokkia tarvitaan tapahtumankäsittelyssä.

- Liittymä käyttöjärjestelmäkohtaiseen koodiin, esimerkiksi C-koodiin, on nyt mahdollista. Tämän tuo mukaanaan uusi ohjelmointirajapinta nimeltään *Java Native Interface.* Liittymä mahdollistaa käyttöjärjestelmäkohtaisen koodin kirjoittamisen ja upottamisen Javan virtuaalikoneeseen.

Muiden parannusten ohella myös Javan suorituskyky on lisääntynyt. Esimerkiksi Javan virtuaalikoneen tärkeimmät osat on nyt kirjoitettu kokonaan konekielellä. Nopeus on parantunut jopa viisin kertaisesti. Luokkien muistitilan vapauttaminen tapahtuu nyt automaattiseti, kun luokkia ei enää tarvita. Java tarvitsee siis entistä vähemmän muistia. Lisäksi Windowsin käyttöjärjestelmäkohtainen liittymä graafiseen rajapintaan (AWT peer-luokat) on kirjoitettu uudelleen. Näin graafisen käyttöjärjestelmän antamia mahdollisuuksia hyödynnetään entistä paremmin ja nopeammin. Lisäksi JDK 1.1 tarjoaa uuden JAR-tiedostomuodon, jonka avulla applettien lataaminen verkosta nopeutuu.

CD:n sisältö

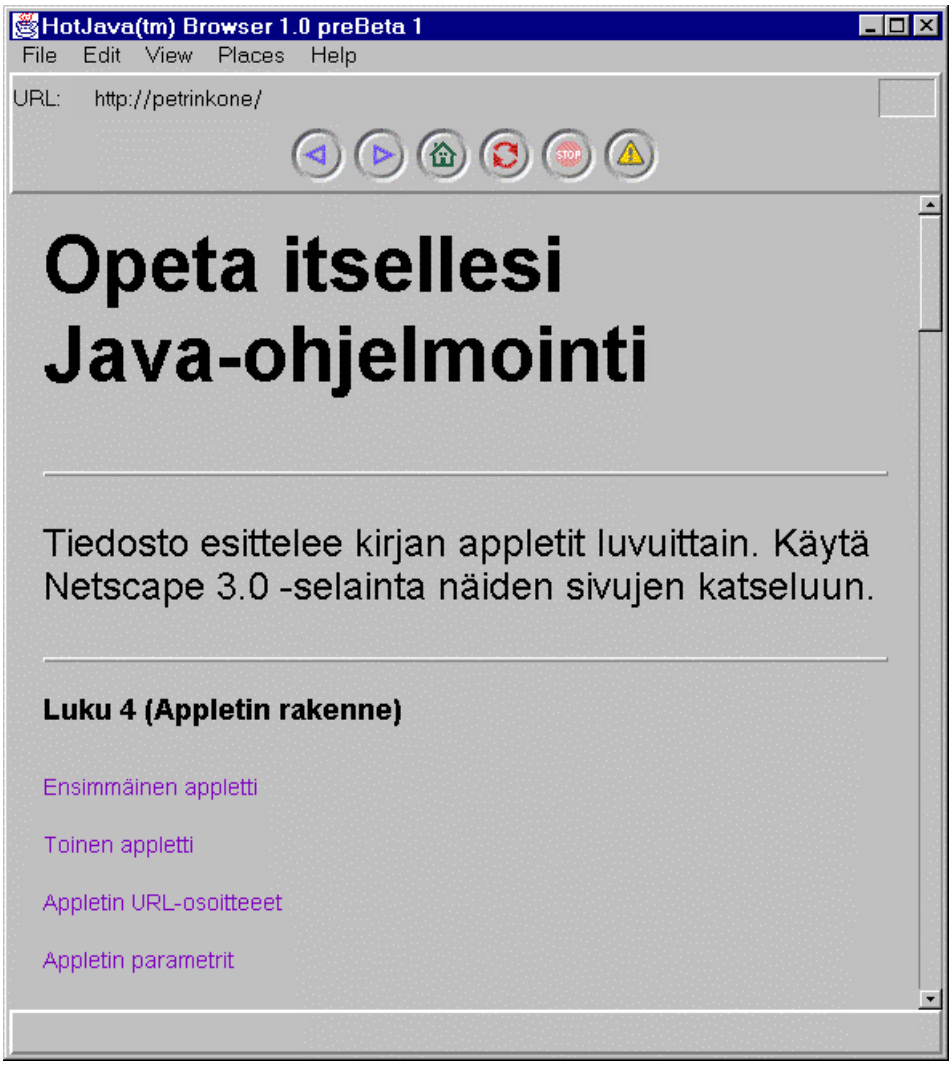
CD:llä on kirjan kaikki appletit (.class), niiden lähdekoodit (.java) ja HTML-sivut (.html). Ne löytyvät hakemistosta \Luokat.

\Luokat	Kirjan appletit, lähdekoodit ja HTML-sivut.

Esimerkiksi appletin EkaApplet nimi levyllä on EkaApplet.class, sen lähdekoodi on tiedostossa EkaApplet.java ja HTML-sivu tiedostossa EkaApplet.html. Esimerkkejä voi selata suoraan CD:ltä, mikäli levyohjain tukee ISO 9660 -standardin mukaisia levyjä. Esimerkiksi DOS:n MSCDEX-ajurin ei tue tätä formaattia oikein.

HUOM! Jos levyohjain ei tunne kyseistä ISO-standardia, levyllä on pakattu tiedosto **luokat.zip**. Se sisältää kokonaisuudessaan luokat-hakemiston sisällön. Tiedosto pitää purkaa kiintolevylle luokat-hakemistoon, jotta index.htm-tiedoston polut pitävät paikkansa.

Esimerkkien katselu aloitetaan selaimessa valikkokomennolla File->Open ja valitsemalla tiedosto **index.htm**, joka sijaitsee CD:n juuressa. Tällä hetkellä vain Hotjava-selaimella voidaan katsoa kirjan esimerkit.



Kirjoittaja

Palautetta kirjasta otetaan mielellään vastaan. Kerro mitkä asiat on selitetty huonosti, mitkä hyvin, mitä muuta kirjassa saisi olla. Ongelmatapauksissa kannattaa kertoa tarkkaan minkälaisessa ympäristössä ongelma esiintyy.

Viihtyisiä lukuhetkiä kirjan parissa.

Petri Kiuttu
petrik@evitech.fi

1. Java-maailma

Tässä luvussa tutustutaan yleisesti Java-maailmaan: Javaa tukeviin WWW-selaimiin, Javan mahdollisuuksiin, turvallisuuteen sekä muun muassa eri ohjelmointiympäristöihin.

1.1 Mitä ovat Java ja JavaScript?

Java on tietoverkkoihin suunnattu ohjelmointikieli. Java-kielellä voi toteuttaa monenlaisia ohjelmia, joita voi ajaa Javaa tukevissa WWW-selaimissa muun muassa Netscapessa. Java-kielisiä ohjelmia, jotka toimivat selaimessa, kutsutaan appleteiksi. Appletit upotetaan HTML-dokumenttiin, ja ne aloittavat toiminsa automaattisesti, kun sivu ladataan. Käyttäjän ei tarvitse siis tietää onko sivulla appletti, eikä applettien vuoksi tarvitse hakea selaimeen ulkoisia apuohjelmia. Appletit toimivat, jos selaimesi tukee Javaa. Kokeile appletteja avaamalla mukana tulleelta CD:ltä tiedosto index.htm ja valitsemalla esimerkiksi linkki Toinen appletti.

Java-kieltä voidaan verrata C++-kieleen, koska ne ovat sukulaiskieliä. C++ on ollut pohjana Javassa, mutta lopputuloksena on kuitenkin hieman erilaiset kielet, koska kielen suunnittelun lähtökohdat ovat olleet erilaiset. Esimerkiksi Java perustuu täysin olioihin toisin kuin C++. Javassa on karsittu C++:n ”huonoja” ominaisuuksia ja tuotu hyviä uusia ominaisuuksia tilalle - ominaisuuksia, jotka sopivat tietoverkkoihin. Näitä ominaisuuksia ovat:

- oliopohjaisuus
- yksinkertaisuus
- siirrettävyys
- tehokkuus
- dynaamisuus
- turvallisuus
- monisäikeisyys
- jaettavuus

Javan oppiminen ja sen täydellinen ymmärtäminen on tietenkin vaikeampaa kuin esimerkiksi HTML-kielen. Java antaa huomattavasti enemmän mahdollisuuksia www-sivujen suunnitteluun kuin HTML voi ikinä antaa. Lisäksi Javalla voi ohjelmoida sovelluksia, jotka toimivat kaikissa käyttöjärjestelmissä. Siksi Javan opettelu on hyvin kannattavaa.

CGI-ohjelmointia harrastava käyttäjä voi ihmetellä miksi Java tarjoaa paremman tavan ohjelmoida interaktiivisia sivuja. Syy on yksinkertainen: CGI ei voi koskaan antaa välitöntä palautetta käyttäjälle. Käyttäjän on ensin painettava jotakin painonappia, ja vasta sitten tulee CGI-ohjelman palaute. Javassa asian laita on toinen, koska ohjelmat voivat reagoida heti käyttäjän antamiin tietoihin.

Netscape Communications ja Sun Microsystems ovat kehittäneet myös helpomman tavan ohjelmoida interaktiivisia sovelluksia WWW:hen. Synnytettiin kieli nimeltä JavaScript. JavaScript ei vastaa laajuudeltaan Javaa, mutta sillä on helpompi toteuttaa ohjelmia, koska HTML-kieltä voidaan hyödyntää yhtenä osana. JavaScript perustuukin täysin HTML-kielen ja JavaScript-kielen yhteiskäyttöön.

1.2 Javaa tukevat selaimet

Tällä hetkellä Javaa tukevat jo useat selaimet. Valitettavasti 1.1-version markkinoilletulo aiheutti sen, että nämä selaimet eivät tue kaikkia mahdollisia Java-appletteja. Vain 1.0-version mukaisia appletteja.

Yleisin ja suosituin www-selain on edelleen Netscape Navigator. Java-tuki on sisältynyt selaimeen versiosta 2 lähtien. Navigatorista on olemassa jo versio 4, tosin se on edelleen testivaiheessa. Kolmos-versiossa on Java- ja JavaScript-kielien virheitä poistettu. Turvallisempaa surffailua on siis luvassa, kun hakee itselleen uusimman version.

Microsoftin Intenet Explorer eli lyhyesti IE on saavuttanut lyhyessä ajassa Netscapen etumatkaa. IE:n suurin etu on sen ilmaisuus. Navigatorin tavoin IE3 tukee Java 1.0 -versiota. Javan lisäksi sekalain tukee Microsoftin esittelemää ActiveX-tekniikkaa, jolla Javan tavoin saadaan paremmin interaktiivisuutta www-sivuille. IE3-selaimen uusimissa versioissa (3.02) on poistettu pahoja tietoturvariskejä. Kannattaa hakea selaimesta aina uusin 3-versio. Nelosversio on jo olemassa, mutta se on pahasti vielä testivaiheessa. IE3-selain on saatavissa osoitteesta www.microsoft.com.

Muita Javaa tukevia selaimia ovat Spyglassin Mosaic-selain ja Oracle PowerBrowser tukevat.

Tämän kirjan kannalta (kesällä -97) paras vaihtoehto on Sunin Hotjava-selain. Selain on edennyt versioon 1.0. Se tukee Javan 1.0- sekä 1.1-versiota. Siksi kaikki CD:llä olevat esimerkit ovat sillä nähtävissä.

- Netscape **Navigator 3.0** tai **4.0**
- Sun Microsystems **Hotjava 1.0**
- Microsoft **Internet Explorer 3.02**
- Spyglass **Mosaic**
- Oracle **PowerBrowser**

Kaikki selaimet on saatavissa Internetistä. Muut ohjelmat ovat ilmaisia paitsi Netscape. Myös Netscapen ilmaisen 30 päivää toimivan demon voi imuroida Internetistä. Windows 95 -käyttäjille hyväksi selaimeksi saattaa tulla Microsoftin Internet Explorer 3.

1.3 Hotjava

Hotjava on Sun Microsystemsin kehittämä WWW-selain, joka kykenee ajamaan verkkoympäristöön suunniteltuja Java-kielisiä ohjelmia eli appletteja. Sunin Hotjava oli ensimmäinen WWW-selain, jossa Java toimii.

Hotjavan ensimmäinen versio perustui alpha-versioon, joka on jo tällä hetkellä on vanhentunut. Javan alkuaikoina yleisenä puheen aiheena WWW-maailmassa olivat Hotjava ja Java, eikä pelkästään Java kuin nyt. **Hang Duke** oli yksi eniten mielenkiintoa herättävistä sovelluksista. Peli tunnetaan paremmin nimellä hirsipuu.

Hotjava on edennyt 1.0-versioon, joka osaa ajaa Javan 1.0- ja 1.1-version mukaisia appletteja. Alpha-version koodi poikkeaa betasta sen verran, että ne eivät ole yhteensopivia. Hotjava on edelleen keskeneräinen tuote monelta osalta, eikä se voi sellaisenaan menestyä. Tuskinpa Sunin tarkoituksena onkaan tehdä siitä kilpailevaa selainta markkinoille. Netscapella on jo vahva markkina-asema ja Microsoft yrittää kuroa sitä kiinni.

Hotjava-selain ei tunne kaikkia HTML-tageja, varsinkaan Netscapen omia laajennuksia. Sen vuoksi sillä ei kannata katsella WWW:ssä sivuja, mutta applettien testaukseen se on omiaan. Hotjava on parantunut paljon alpha-versiosta, joten se on ainakin tutustumisen arvoihin, vaikka sitä ei muuten käyttäisikään.

Hotjavan erikoisuus on se, että se on kirjoitettu kokonaan Java-kielellä. Käyttäjälle Hotjava on kuitenkin aivan tavallinen selain, kuten Netscape tai Mosaic.

1.4 Javan ja JavaScriptin mahdollisuudet

HTML-kielen avulla voidaan tuottaa helposti www-sivuille tekstiä erilaisilla kirjasinlajeilla, sekä interaktiivisia elementtejä painonapeilla ja listoilla yhdessä CGI:n kanssa. Niin hyvä kuin HTML-kieli onkin, se ei tarjoa ominaisuuksia välittömään kaksisuuntaiseen interaktiivisuuteen.

Java-kielen avulla WWW-sivuille voidaan tehdä mitä erilaisimpia sovelluksia. On mahdollista ohjelmoida esimerkiksi pelejä käyttäjien viihdyttämiseksi. Esimerkiksi Java-ohjelmointipaketin mukanakin tulee ristinolla (TicTacToe). Javan yksi parhaimmista ominaisuuksia on animaatioesitykset. Kotisivulleen voi nyt tuottaa mielekkään sarjakuvan peräkkäisillä kuvilla, esimerkiksi kävelevän miehen tai nyökyttelevän oman kuvan. Animaatio voidaan toteuttaa ohjelmoitavuuden ansiosta muutamalla kilotavulla (riippuen tietenkin kuvien koosta), kun

vastaavanlainen esitys videokuvana kuluttaisi useita kymmeniä kilotavuja. Pelien ja animaatioiden lisäksi Javalla voi ohjelmoida hyötyohjelmia, demoja, scrollereita ja muita mitä hienoimpia sovelluksia. Käyttäjien kotisivut saavat näin uusia ulottuvaisuuksia, ja selaamisestakin tulee taas entistä mielekkäämpää. Java-ohjelmointipaketin mukana tulee kymmenkunta erilaista sovellusta, joista voi poimia ideoita ja mallia ohjelmointiin.

Sun on kehittämässä prosessoreita, joiden konekielenä toimii Java. Tällaiset prosessorit ovat tietenkin erittäin nopeita suorittamaan Java-koodia. Prosessoreita tullaan käyttämään matkapuhelimissa, viihde-elektroniikassa ja verkkotietokoneissa. Nyt jo useat yritykset ovat ilmoittaneet ottavansa Javan laitteisiinsa. Esimerkiksi suomalainen Nokia käyttää Javaa matkapuhelimissa.

1.5 Turvallisuus

Java-kielen tultua WWW-maailmaan alettiin keskustella applettien turvallisuudesta. Onhan selvää, että maailmassa on monenlaisia ohjelmoijia, jotka mielellään käyttävät Javaa vääränlaisiin tarkoituksiin. Java-kieli on kuitenkin alun alkaen suunniteltu verkkoihin, ja siksi sen ominaisuudet ovat hyvät.

Käyttäjälle tulee tietenkin mieleen virukset. Viruksia on Internet pullollaan, joten yhtä hyvin virukset voisivat huomaamattomasti siirtyä Internetissä applettien mukana. Niiden vaara on onneksi eliminoitu, koska Javassa ei ole mahdollisuutta kirjoittaa suoraan keskusmuistiin tai kiintolevylle. Javassa on muun muassa sisäänrakennettu tarkastin, joka tutkii jokaisen appletin oikeellisuuden ennen ajamista. Virheellisiä (epämääräisiä käskyjä) sisältäviä appletteja ei ajeta lainkaan.

Valittevasti pitää todeta, että mikään ei ole täydellistä. Javan kohdalla on jo ilmennyt jonkin asteisia viruksia. On pystytty ohjelmoimaan appletteja, jotka ovat vioittaneet kiintolevyn sisältöä. Lisäksi on tehty applet, joka sotkee Javan turvallisuusmekanismin (*Java Applet Security Manager*) täysin. Hyökkääjä-applet kytkeytyy uhrin koneeseen, kun uhri on ensin ladannut ”viattoman” appletin. Hyökkääjä-applet suorittaa hyökkäyksen itse uhriin tai kaikkiin sen alla oleviin koneisiin.

JavaScript-kielikään ei ole säästynyt bugeilta. Eräs bugi sallii salaisten tietojen, kuten käyttäjän email-osoitteen, lähettämisen eteenpäin käyttäjän tietämättä. JavaScriptin tapauksessa Netscapen Security Preferences -dialogin asetuksilla ei ole merkitystä versiossa 2. Jos pelkää näitä bugeja, ainoana vaihtoehtona on käyttää selainta, joka ei tue Javaa.

1.6 Kehitysympäristöt

Javan kohdalla tunnetuin ja ilmainen kehitysympäristö on tietenkin **JDK** (*Java Deveploper’s Kit.* JDK sisältää runsaasti työkaluja Java-ohjelmointiin. Liittessä A kerrotaan niistä lisää.

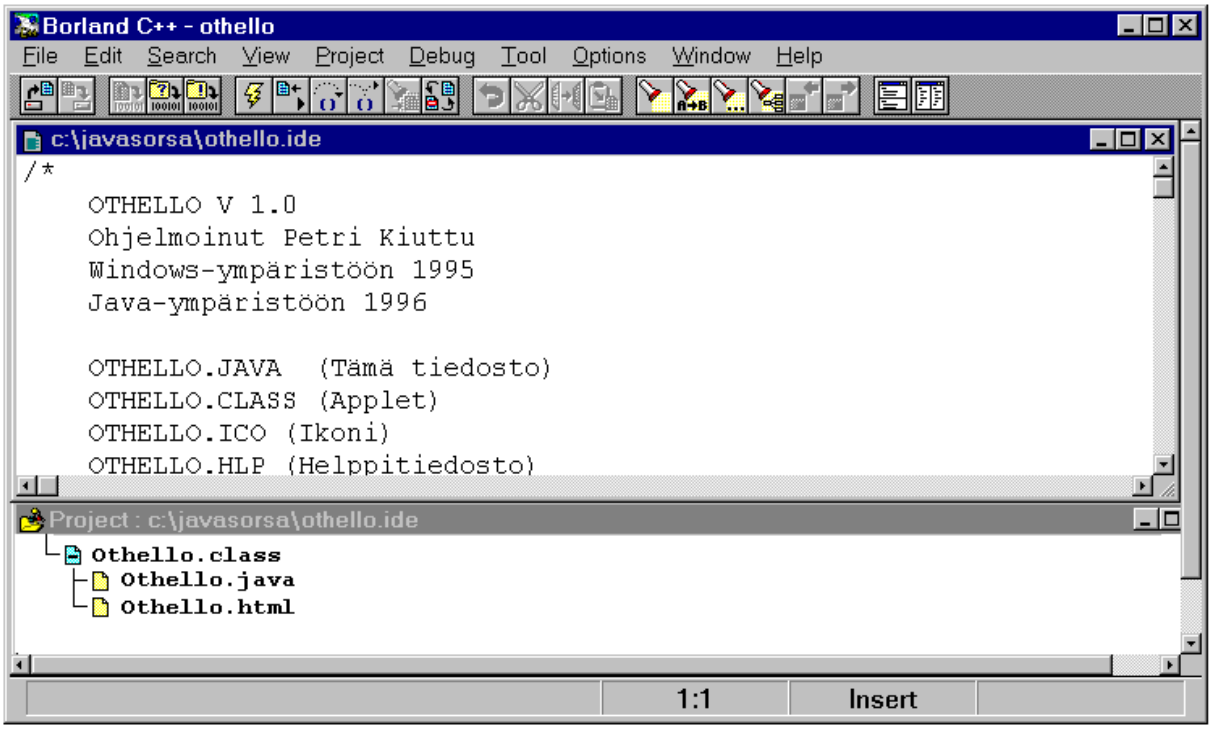
Nykyinen ohjelmointipaketti on JDK1.1.2. Ohjelmointipaketti sisältää kääntäjän, jolla Java-ohjelmia käännetään tavukoodeiksi sekä Java-tulkin sovellusten ajamiseen ja appletviewer-ohjelman, jolla testataan appletteja. Tietenkin ohjelmointipaketissa on tarvittavat luokat ja niiden funktiot ohjelmoinnin aloittamiseen.

BIN	Ohjelmat (javac, appletviewer)
LIB	DLL-kirjastot
DEMO	Esimerkkiohjelmia
INCLUDE	Include-tiedostoja
DOCS	Ohjelmointirajapinta

Sunin JDK ei ole siinä mielessä hyvä, että siinä ei ole integroitua kehitysympäristöä. Ongelman voi kiertää hakemalla Internetistä vapaalevitteisen, ohjelmointiin tarkoitetun editorin. Editorin avulla pystyy luomaan itselleen sopivan kehitysympäristön lähdekoodin muokkaukseen (mahdollinen värikorostus), kääntämiseen ja applettien testaukseen (Java-tulkilla). Tällaisia editoreja Windowsiin ovat muun muassa Zeus, Brief ja PFE. Editoreista kannattaa ottaa 32-bittinen versio, jotta pitkien tiedostonnimien käyttö olisi mahdollista. Muussa tapauksessa syntyy hankalia ongelmia, koska JDK:n javac (*Java Compiler*) ei suostu toimimaan vanhalla 8.3-tiedostojärjestelmällä.

Borland C++ 5.0 Development Suite (Windows) eli Borlandin C++-kehitysympäristö tukee nyt myös Javaa (versiota 1.0). Samalla tavalla kuin ohjelmaa käytetään C++-koodin tekoon DOS- tai Windows-käyttöjärjestelmiin, sillä voidaan tuottaa Java-koodia. Ohjelma vaatii käyttäjältään Javan kieliopin tuntemusta, joten pelkän C++:n taitaminen ei riitä applettien tuottamiseen. Borlandin kehitysympäristö on kuitenkin yksi parhaimpia, ja onhan se oiva etu Borlandiin tottuneelle C++-ohjelmoijalle, kun ei tarvitse opetella uusia kehitysympäristöjä. Borland C++ 5.0 on kaupallinen tuote.

Borland on myös aktiivisesti Javan kehitystyössä mukana. Yhtiö lupaileekin oman JDK:n (IDE),



jossa on mukana hyvä debuggeri ohjelman testaukseen, editori (Javan kieliopille sopiva rivien värikorostus sekä tietenkin kääntäjä ja appletin testausmahdollisuus). Lisäksi Borland on kehittänyt **AppAccelerator** nimisen JIT-kääntäjän, josta Netscape on erittäin kiinnostunut. Kääntäjä tullaan luultavasti näkemään vielä jatkossakin Navigatorin uusimmissa versioissa. Luvassa on siis nopeutta applettien suorittamiseen.

1.7 Sovelluskehittimet

Javaa varten on kehitetty useita sovelluskehittimiä, joilla applettien tekeminen sujuu helpommin. Sovelluskehittimet ovat hyviä niille, jotka eivät ole kiinnostuneet itse Java-kielestä, vaan haluavat nopeasti ja helpolla tavalla tuottaa appletteja omille WWW-sivuilleen. Kehittimet ovat WYSIWYG-tilassa eli käyttäjä näkee lopullisen tuloksen heti.

- Microsoft **Visual J++**
- Borland **JBuilder**
- Powersoft **Optima++**
- Symantec **Cafe**
- Sun **Java Workshop**
- RoqueWave **Jfactory**
- FutureTense **Texture**

- Step Ahead Software **Javelin**

Sovelluskehittimiä on nyt useita, ja lisää tulee jatkuvasti. Nähtäväksi jää, saavuttaako jokin kehittimistä ylivoimaisen ykköstilan. Mielenkiintoisia kehittimiä ovat muuan muassa Sunin Java Workshop (kirjoitettu Javalla), Visual J++ sekä Symatec Cafe. Visual J++ -ohjelmasta voi hakea ilmaisen version internetistä. Visual J++ on lisäksi sisällytetty Microsoftin ohjelmointipakettiin. Microsoftin selaimessa on lisäksi JIT-kääntäjä Javalle.

Sovelluskehittimien haittana on tietenkin niiden kaupallisuus. Ohjelmat maksavat yleensä 1000 markasta ylöspäin. Java-ohjelmointikieltä tunteva pystyy nyt ja tulevaisuudessa luomaan applettinsa ilmaiseksi Sunin JDK:n avulla.

2. Johdatus Java-kieleen

Tässä luvussa tutustutaan päällisin puolin Java-kieleen. Luvun tarkoituksena on esitellä Javan ominaisuuksia ja tekniikkaa. Tämä luku toimii siten perustana kirjan seuraavalle luvulle, jossa aloitetaan Java-kielen opiskelu.

2.1 Ominaisuudet

Java on olio-ohjelmointikieli. Java on tyypiltään tulkattavaa (vrt. basic), joten ohjelman sisältämät tavukoodit muutetaan konekieliseksi ohjelman suorittamisen aikana. Tulkkaamisella saavutetaan se etu, että ohjelmat ovat käyttöjärjestelmästä riippumattomia. HTML-sivuilla olevat ohjelmat ovat ajettavissa Windows-, Solaris- ja Macintosh-käyttöjärjestelmissä tai missä tahansa muissa Javaa tukevissa käyttöjärjestelmissä. Java tukea ei ole vielä sisällytetty kaikkiin käyttöjärjestelmiin, mutta tarkoitus on saada Java toimimaan mahdollisimman monessa järjestelmässä.

Käyttöjärjestelmästä riippumaton kieli on siirrettävää. Tämä onkin yksi Javan tärkeimpiä ominaisuuksia. Java-ohjelmoijien ei tarvitse välittää käyttöjärjestelmäkohtaisista eroista. Esimerkiksi int-tietotyypin (kokonaisluku) varaama tila on aina sama, oli kyseessä mikä käyttöjärjestelmä tahansa. Riippumattomuus käyttöjärjestelmästä näkyy myös ohjelmointirajapinnassa (*API*), joka tarjoaa erittäin korkeantason luokat ja funktiot ohjelmoijille.

Java-kieli on myös turvallista, koska se ei salli ohjelmoijien kirjoittaa epämääräistä koodia. Javassa ei ole esimerkiksi mahdollista käyttää suoria muistiinosoituksia - virusten vaara on täten eliminoitu mahdollisimman pieneksi. Tietoturvamielessä Javan tärkeät ominaisuudet ovat:

- Osoittimien käyttö kielletty
- Varattujen muistialueiden ulkopuolelle ei voi osoittaa
- Tyyppimuunnokset tarkistetaan
- Käännösvaiheessa tarkistetaan kaikkien funktiokutsujen oikeellisuus
- varattu muisti vapautetaan automaattisesti

Javassa on varma muistinhallintatekniikka. Varattua muistia ei tarvitse vapauttaa, koska on olemassa automaattinen vapautusmekanismi. Varatun muistialueen ulkopuolelle ei voi kirjoittaa: osoittimien käyttö on kielletty ja taulukkojen yhteydessä tarkistetaan aina, mitä alkiota osoitetaan.

Javassa on kaksinkertainen virheidenkartoitusmekanismi: ensiksi käännösvaiheessa tarkistetaan kaikki mahdolliset virheet, kuten funktiokutsujen oikeellisuus. Toisen kerran virheet eliminoidaan ajon aikana. Harvoin selain kaatuu Java-ohjelman virheisiin.

2.2 Ohjelmointi

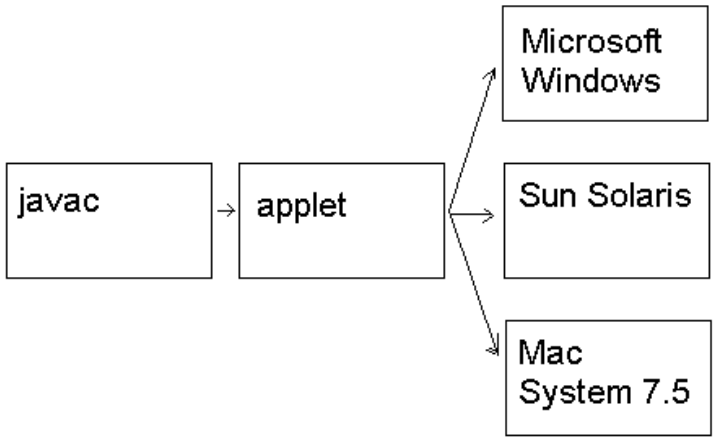
Java-ohjelma rakennetaan yksinomaan luokista ja niiden sisältämistä funktioista eli metodeista. Ohjelmointi tapahtuu käyttäen Javan APIa (*Application Programmer Interface*) eli ohjelmointirajapintaa.

Java-kielinen ohjelma perustuu täysin olioihin. Ei ole olemassa yhtäkään Java-ohjelmaa, joka on tehty ilman olioita! Javaa verrataan yleensä C++-kieleen, koska se osaltaan perustuu siihen. Kielien ero on kuitenkin merkittävä siinä suhteessa, että C++-kieli perustuu C-kieleen. Vasta C++-kielisen ohjelman sisällä voidaan määritellä C++:n erikoisominaisuuksia, kuten olioita. C++-kieleen on vain lisätty olio-ominaisuudet, joten se ei ole täysiverinen olio-ohjelmointikieli suhteessa Javaan.

Windows-ohjelmointia tunteville kerrottakoon, että Java- ja Windows-ohjelmoinnin perusperiaatteet ovat samanlaisia. Jos osaa ohjelmoida Windowsille, osaa pienen opettelun jälkeen ohjelmoida Javalla. Java-kieleen siirtyminen on myös helppoa C++-kieltä tuntevile, koska Java-kieli on C++-kielen tapaista. Opettelun kohteeksi jäävät uudet funktiot, jotka ovat nopeasti opittavissa.

2.3 Kirjoita kerran, aja missä vain

Java on nyt muuttumassa niin, että sitä ei ole tarkoitettu pelkästään WWW-teknologian kehittämiseen. Javasta tehdään ohjelmointikieli siinä määrin kuin C++-kielikin on. Tämä tarkoittaa sitä, että käyttöjärjestelmissä on sisäänrakennettu Java-tuki. Ei tarvita siis WWW-selainta, joka osaa ajaa Java-ohjelmia, vaan ohjelmat voidaan ajaa suoraan tietokoneen käyttöjärjestelmässä aivan kuin mikä tahansa ohjelma. Eikä tässä vielä kaikki. Koska Java on siirrettävää, *Java-ohjelmat voidaan ajaa missä laitteisto- ja käyttöjärjestelmässä tahansa!* Kaikista koneista tulee näin yhteensopivia. Kun osaa ohjelmoida Javalla, voi tehdä sovelluksia kaikkiin eri käyttöjärjestelmiin sekä tietokoneisiin!



Nyt on siis viimeistään kaikkien C++-ohjelmoijien syytä herätä! On selvää, että tulevaisuudessa ei tarvita C++-ohjelmoijia jokaiseen tehtävään: ei ole enää Unix C++ -asiantuntijaa, eikä Visual Basic -ohjelmoijaa. On selvää, että yritykset alkavat pian koota Java-ohjelmointitiimejä. Yrityksissä on järkevämpää käyttää Javaa kuin C++-kieltä, koska säästetään kustannuksissa. Monet ovat sitä mieltä, että Java ei tule syrjäyttämään koskaan C++-kieltä, koska Java on hitaampaa suorittaa tulkkaamisen vuoksi. Tämän hetken tekniikalla, jossa tavukoodit tulkitsee Java Virtual Machine, koodin suorittaminen on vielä hidasta. Mutta lopulta, kun Java on suoraan käyttöjärjestelmässä ja käytetään JIT-tekniikkaa, päästään yhtä nopeaan koodin suorittamiseen kuin C++:ssa.

Internet-ohjelmointi on tähän mennessä ollut käyttöjärjestelmäkohtaista, poislukien HTML-kieli. Internetiä hyödyntävät ohjelmat, kuten WWW-selain, on tehty tietylle käyttöjärjestelmälle, esimerkiksi Windowsille. WWW:ssä toimiva CGI on myös käyttöjärjestelmäkohtaista, ja sen koodi suoritetaan siinä palvelimessa, johon ollaan yhteydessä. Java tuo nyt tähän tilanteeseen aivan uuden ratkaisun. Koska Java on siirrettävä, laitteisto- ja käyttöjärjestelmäriippumaton ohjelmointikieli, ohjelmoija voi keskittyä täysin ohjelman sisältöön. Konekohtaisuudesta ei tarvitse välittää.

2.4 JavaScript-kieli

JavaScript on Java-kielestä kehitetty olio-pohjainen ohjelmointikieli, jonka tarkoituksena on helpottaa interaktiivisten multimediasovellusten kehittelyä. Varsinaisen helpotuksen tuo mukanaan HTML-kielen ja JavaScript-kielen yhteiskäyttö. Javassa ei ole esimerkiksi mahdollisuutta mitenkään hyödyntää HTML-kielen helppoutta luoda graafisia elementtejä (painonappeja, valintalistoja jne.) JavaScriptissä elementtien luonti onnistuu helposti HTML-kielen tageilla ja elementit linkitetään osaksi ohjelmaa. JavaScript-ohjelmat pystyvät vastaamaan käyttäjän tekemiin tapahtumiin, kuten hiiren napautuksiin ja sivun valintaan. HTML-kielisiin ohjelmiin saadaan näin "älykkyyttä", koska esimerkiksi elementtien sisältämä informaatio voidaan lukea ja palauttaa heti informaatiota käyttäjälle. JavaScript tuo käyttäjille välittömän interaktiivisuuden samoin kuin Java.

JavaScript-kielinen ohjelma on varsinaisesti osa HTML-dokumenttia, ohjelman lähdekoodi on siten nähtävissä selaimien valikkokomennolla View->Source. JavaScript muistuttaa Java-kieltä, mutta siitä puuttuu vahva tyypitys, joka Java-kielessä on olennaista. Kieli tukee myös funktioita, kuten Java, mutta ilman funktioiden määrittelyä. Tyypityksen ja funktioiden määrittelyn pois jättäminen alentaa samalla ohjelmointikynnystä - lähes kuka tahansa voi kirjoittaa JavaScript-kielisiä ohjelmia. JavaScriptistä kiinnostuneiden kannattaa tutustua ATK-kustannuksen kirjaan Opeta itsellesi JavaScript.

2.5 Java-sovellukset

Java-sovellukset ovat niin kutsuttuja *Stand alone* eli yksistään toimivia ohjelmia. Tällaiset sovellukset voidaan käynnistää Java-tulkin päällä.

Java-tulkki vaatii, että lähdekoodissa **`main`**-funktio; ellei ole, ei myöskään sovellus käynnisty. Selaimet eivät tunne, eivätkä käytä main-funktiota lainkaan. Sen vuoksi ne tarvitsevat oman erilaisen liityntätavan ohjelmaan. Siitä kerrotaan lisää luvussa 5.

2.6 Ensimmäinen Java-sovellus

Olio-ohjelmoinnin perusteisiin perehdytään tarkemmin vasta seuraavassa luvussa, mutta tässä vaiheessa katsotaan kuinka muodostetaan yksinkertaisin Java-sovellus. Tätä sovellusta ei kannata yrittää ajaa WWW-selaimessa, koska se ei siinä toimi!

```
public class EkaOhjelma
{
  static public void main (String arg[])
  {
   System.out.println(”Hello World”);
  }
}
```

Ohjelma alkaa rivillä, joka kertoo että muodostetaan yleinen luokka nimeltään EkaOhjelma.

```
public class EkaOhjelma
```

Toinen rivi ohjelmassa kertoo, että luokassa on main-funktio, joka on yleinen ja staattinen (niiden merkitykseen palataan luvussa 3). Main-funktiosta aloitetaan ohjelman suorittaminen. Funktiolle välitetään komentoriviltä annetut parametrit arg-nimiseen merkkijonoon (=olioon).

```
static public void main (String arg[])
```

Kolmas rivi ohjelmassa tulostaa tekstin ”Hello World”. Tulostaminen tapahtuu luokkakirjaston luokalla **java.lang.System**. Kirjasto ladataan automaattisesti jokaiseen Java-ohjelmaan.

```
System.out.println(”Hello World”);
```

Ohjelmointi Java- ja C++-kielillä perustuvut luokkakirjastoihin. Luokkakirjastot ovat rajapinta ohjelman ja käyttöjärjestelmän välillä. Kummatkin kielet ovat korkean tason kieliä, joten tietokoneen laitteistokokoonpanoa tai käyttöjärjestelmää ei tarvitse tuntea.

Luokkakirjastot sisältävät luokkia (*class*). Luokat puolestaan sisältävät funktioita. Luokassa olevien funktioiden virallinen nimitys jäsenfunktio tai metodi (*method*). Tässä kirjassa kuitenkin puhutaan vain funktioista.

```
funktio();          // C:n funktiokutsu
olio.funktio();     // Javan metodikutsu
```

2.7 Lähdekoodin kääntäminen JDK:lla

Lähdekoodi voidaan kirjoittaa millä tahansa editorilla, joka kykenee tallentamaan ASCII-muotoista tekstiä. Tällaisia editoreja ovat muun muassa Muistio (Windows), pico (Unix) ja Teksturi (Mac). Tiedosto tallennetaan **.java**-päätteisenä.

- Lähdekoodi kirjoitetaan ASCII-muodossa
- Tiedosto tallennetaan .java-päätteisenä

Kääntämisen lähdekoodista tavukoodeiksi eli tokeneiksi suorittaa ohjelma nimeltä **javac** (*Java Compiler*). Se löytyy JDK:n hakemistosta **\java\bin**. Hakemistoon kannattaa laittaa hakupolku, koska kääntäjää tarvitaan useasti.

```
set PATH = %PATH%;C:\JAVA\BIN;
```

Kääntäminen voidaan suorittaa myös muissa hakemistoissa kuin \java\bin-hakemistossa, jos ympäristömuuttujat on asetettu oikein. Velho asettaa ympäristömuuttujat, joten useinkaan näitä ympäristömuuttujia ei tarvitse laittaa käsin.

```
set HOME = c:\          (riittää DOS-koneissa yksinään)
set HOMEPATH = \
set HOMEDRIVE = c:
```

Javac-kääntäjän voi käynnistää kirjoittamalla komentoriville komennon **javac** ja sille parametriksi tiedoston nimen. Lähdekooditiedoston nimen on päätyttävä **.java**-päätteeseen, koska muut päätteet javac hylkää.

javac EkaOhjelma.java

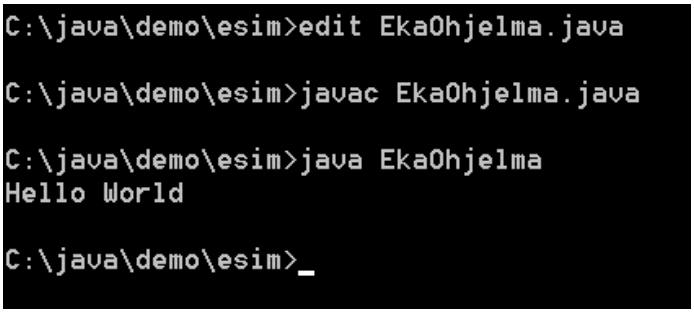
Lopputuloksena, kun käännös lopulta onnistuu, on tavukoodeista muodostuva luokka. Javassa ei tarvitse ajaa enää erillistä link-ohjelmaa, kuten C++-kielessä.

EkaOhjelma.class

Nyt ensimmäinen ohjelma voidaan ajaa Java-tulkilla (mikäli käyttöjärjestelmäsi ei tue suoraan Javaa). Java-tulkki löytyy samasta hakemistosta kuin Java-kääntäjäkin. Se käynnistyy käskyllä **java**, sille annetaan parametriksi Java-ohjelman nimi ilman .java-päätettä.

java EkaOhjelma

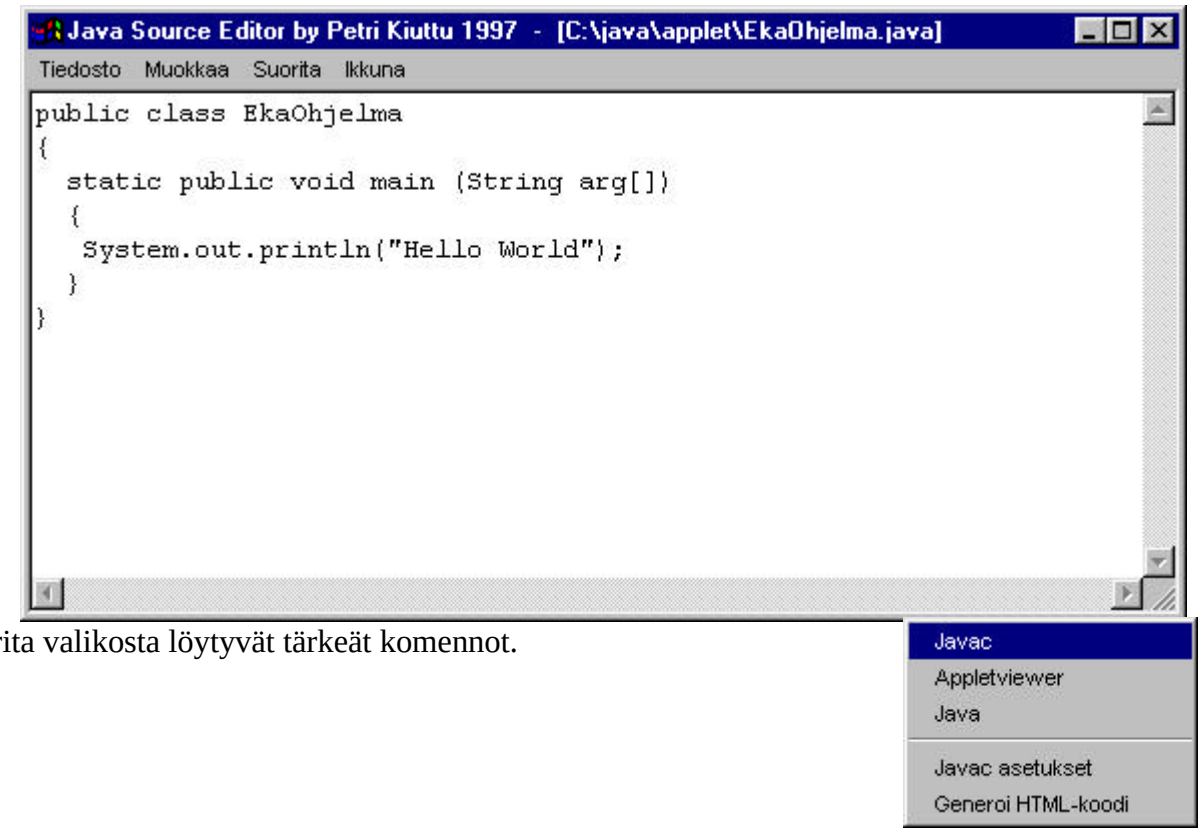
Kun nämä toimenpiteet suoritetaan ensimmäiselle Java-ohjelmalle (EkaOhjelma), saadaan tulokseksi:



2.8 Editori

Editori on kirjan CD:llä mukana tullut Java-ohjelmointiin tarkoitettu tekstieditori. Se on toteutettu Java-kielellä ja toimii sen vuoksi jokaisessa käyttöjärjestelmässä. Editori-ohjelman tekstinmuokkaus on toteutettu natiivilla TextArea-komponentilla. Siksi editorin ominaisuudet ovat riippuvaisia käyttöjärjestelmästä (nopeus, tekstinmuokkaustoiminnot, merkkien määrä).

Editori näyttää normaalilta Windows-ohjelmalta. Käyttöliittymä on esitetty alla. Editorissa on valikot Tiedosto, Muokkaa, Suorita ja Ikkuna. Tiedosto-valikossa on tyypilliset tiedoston avaa- ja tallenna-komennot sekä tulosta-komento. Muokkaa valikossa on tavanomaiset leikkaa-, liimaa- ja liitä-komennot.



Suorita valikosta löytyvät tärkeät komennot.

Javac	Kääntää lähdekoodin tavukoodiksi javac-kääntäjällä.
Appletviewer	Ajaa www-sivun, johon appletti on sijoitettu
Java	Käynnistää Java-tulkin
Javac asetukset	Käyttäjä voi antaa parametrit javac-kääntäjälle tämän komennon avulla.
Generoi HTML-koodi	Tuottaa automaattisesti www-sivun, jota tarvitaan appletin ajamiseksi.

Kun lähdekoodi on käännetty, mahdolliset virheilmoitukset tulostuvat toiseen ikkunaan. Tämä ikkuna avautuu automaattisesti editoriin. Takaisin lähdekoodi-ikkunaan pääsee näppäimellä CTRL-1 tai Ikkuna-valikon komennolla Lähdekoodi.

Editori-ohjelma sijaitsee CD:llä hakemistossa \Luokat. Sen voi ajaa Java-tulkilla.

3. Java-kielen perusteet

Tässä luvussa tutustutaan Java-kielen kielioppiin. Lukijalla ei tarvitse olla kokemusta C++ eikä C-kielestä, mutta siitä on hyötyä. Jos lukijalla ei ole kokemusta näistä kielistä, suosittelen tutustumista seuraaviin kirjoihin:

Opeta itsellesi C++

Olio-ohjelmoinnin perusteet

Pieni oliokirja

Java-kielen periaatteen oppii nopeasti tästä luvusta, tosin aivan perusteellinen Java-kielen opas tämä luku ei ole. Perusteelliseen oppaaseen, jossa opastettaisiin monin esimerkein kaikki mahdolliset asiat, tarvittaisiin ainakin sama määrä sivuja kuin tähän kirjaan on varattu.

3.1 Java- ja C++-kielen eroja

Java-ohjelmointi on samankaltaista kuin C++-ohjelmointi. Java on lausekieli kuten C:kin, eli lähdekielinen ohjelmakoodi muodostuu lausekkeista ja lauseista. Kielillä on paljon muitakin yhtäläisyyksiä, kuten tietotyypit, jotka ovat nimiltään samanlaisia. Aivan samanlaisia ne eivät kuitenkaan ole!

Javassa ei ole lainkaan C:stä tuttuja struktuureita eikä unioneita. Nämä on tietenkin korvattavissa luokilla. Define-lauseita, joilla voidaan määritellä lukuarvoja nimille, ei Java myöskään sisällä. Ne eivät ole korvattavissa. Osoittimien puuttuminen estää muistin väärinkäytön. Moniperiytyvyyttä ei ole olemassa C++-kielen tavoin, mutta Java mahdollistaa sen omalla tavallaan. Funktioiden pitää olla aina luokissa, eli käytännössä funktiot ovat aina metodeita eli jäsenfunktioita. Tässä kirjassa käytetään nimitystä funktio metodin sijasta. Ylikuormittaminen Javassa on mahdollista vain funktioille, ei operaattoreille. Seuraavaan taulukkoon on koottu kielien eroja.

C++-kieli	**Java-kieli**
Struktuurit, unionit ja oliot	Oliot
Symboliset vakiot (define)	Ei symbolisia vakioita
Osoitin muuttujat	Ei osoitinmuuttujia
Moninkertainen periytyvyys	Periytyvyys
Funktiot ja metodit	Metodit
Typedef-lause	Ei typedef-lausetta
Muuttujien muistinvarausluokat	Muunnin
Keinotekoiset boolean-muuttujat	Oikeat boolean-muuttujat
Bittikentät	Ei bittikenttiä
Break-, continue- ja goto-lauseet	Break- continue- ja loop-lauseet

Funktioiden protyyppi Operaattorien ylikuormitus	Ei prototyyppejä Ei operaattorien ylikuormitusta

Java-ohjelmointi on kuin Windows-ohjelmointia - useat Windowsin-ohjelmointirajapinnan funktioiden nimet ovat samannimisiä kuin Javan funktioiden. Javassa tosin on jaettu funktiot useampaan luokkakirjastoon kuin Windowsissa. Muita yhtäläisyyksiä ovat takaisinkutsuttavat funktiot, joiden tarkoituksena on informoida ohjelmaa jostakin tapahtumasta. Windows- ja Java-ohjelmoinnin suurimman eroavuuden aiheuttaa Javan "pakollinen" olio-ohjelmointi.

3.2 Java-kielen kirjoitusasu

C++-kielen tavoin Java-kieli sisältää kommentteja, lauseita ja funktiokutsuja. Ohjelman suoritettava osa koostuu lauseista. Lausetta voidaan verrata käskyyn, joka ohjaa tietokoneen toimintaa. Lauseet puolestaan sisältävät yhden tai useamman lausekkeen. Lauseke on muuttuja, operaattori tai funktiokutsu, jolla on jokin arvo. Seuraavassa on esimerkkejä yksinkertaisista lausekkeista.

```
1+1
update()
```

Lauseella ei ole arvoa ja se päättyy **aina** puolipisteeseen. Lause voi olla esimerkiksi seuraavanlainen.

```
a = 1 + 1;
update();
```

Javassa käytetään aaltosulkeita sitomaan useita lauseita yhteen. Esimerkiksi if-lauseen vaikutusaluetta voidaan kasvattaa aaltosulkeilla osoittamaan useita lauseita, kun se normaalisti vaikuttaa vain seuraavaan lauseeseen.

```
if (a == 1)
{
  a = 2;
  b = 2;
}
```

Aaltosulkeiden sijoittelun ei tarvitse olla yllä olevan kaltainen. Esimerkiksi useissa Java-ohjelmointipaketin esimerkeissä käytetään seuraavanlaista sijoittelutapaa. Aaltosulkeiden ja

sisennyksen merkitys on vain visuaalinen, eikä sillä ole merkitystä ohjelman suorituksen kannalta.

```
if (a == 1) {
  a = 2;
  b = 2; }
```

Java-kieli on ulkoasultaan hyvin samanlaista kuin C++-kieli. C++-kieltä tunteva voikin lukea melko sujuvasti Java-kielellä tehtyjä ohjelmia. Java-kieli on tyyliltään hierarkinen, joten ylimääräisien välilyöntien ja tyhjien rivien käyttäminen on suositeltavaa. Se on hyvä tapa selkeyttää lähdekoodien luettavuutta.

3.3 Kommentit

Kommentit selventävät lukijalle tehokkaasti mitä lähdekoodissa tehdään milläkin kohtaa. Kommentteja on hyvä käyttää, jo pelkästään oman itsensä vuoksi, jotta myöhemminkin ymmärtää ohjelmarivien merkityksen. Kommenttien toinen merkitys on kuvata lähdekooditiedoston alussa, mitä ohjelma tai luokka tekee, kuka on lähdekoodin kirjoittaja, milloin koodi on kirjoitettu, mikä on versionumero jne.

Java-kieli tuntee kahdenlaisia kommentteja. Ne ovat:

```
// Tämä on kommentti
```

Ja

```
/* Tämäkin on
    kommentti */
```

Kommenttien ero on se, että ensimmäistä käytetään kommentoimaan yksittäisiä rivejä ja jälkimmäistä useampia rivejä.

```
/*
   Ohjelma: testi
   Tiedosto: testi.java
   Kuvaus: Esittelee kommentointia
   Versio: 1.0  (15.4.96)
*/
public class testi
{
```

```
 public static main(String args[])
 {
   //System.out.println(”Tämä ei tulostu”);
   System.out.println(”Tämä rivi tulostuu näytölle”);
 }
}
```

Tulos:

```
    Tämä rivi tulostuu näytölle
```

3.4 Muuttujat ja tietotyypit

Muuttujaan voidaan varastoida lukuarvoja ja lukuarvo voidaan myöhemmin lukea sieltä. Muuttujaan voidaan sijoittaa lukuja esimerkiksi lauseella `a = 1 + 1`. Tässä **a** on muuttuja ja 1+1 on arvo, joka tähän muuttujaan tallennetaan. Muuttujan arvo voidaan lukea muuttujasta milloin tahansa esimerkiksi lauseella `b = a;`.

```
public class testi
{
 public static void main(String args[])
 {
  int a = 1 + 1;
  System.out.println("A:n arvo on "+a);
 }
}
```

Tulos:

```
    A:n arvo on 2
```

Muuttujan nimi voi olla mikä tahansa merkki tai merkkiyhdistelmä, joka ei sisällä laittomia kirjaimia.

```
    int esimerkki = 6;
    int muuttuja = 2;
```

Muuttujien nimen on alettava kirjaimella (a-z tai A-Z), alaviivalla tai dollarimerkillä. Nimessä saa esiintyä myös numeroita, mutta ei välilyöntejä. Esimerkiksi seuraavat muuttujien nimet ovat laillisia. Huomaa, että isot ja pienet kirjaimet tulkitaan eri kirjaimiksi. Seuraavat muuttujat ovat siten eri nimisiä.

```
hotjava10A = 1;
Hotjava10a = 2;
```

Laiton muuttuja on esimerkiksi varattu sana if. Eli määrittely `if = 2` tuottaa virheilmoituksen.

```
if = 2;          // VÄÄRIN!!!
```

Muuttujan nimi saa sisältää sanan if, mikäli se ei esiinny yksinään.

```
int ifa;         // OIKEIN!!!
```

Seuraavana on esitetty Java-kielen varatut sanat, jotka eivät voi olla sellaisenaan muuttujien niminä. Taulukossa on esitetty myös goto varatuksi sanaksi, vaikka se ei ole käytössä.

abstract	**default**	**goto**	**null**	**synchronized**
boolean	**do**	**if**	**package**	**this**
break	**double**	**implements**	**private**	**threadsafe**
byte	**else**	**import**	**protected**	**throw**
byvalue	**extends**	**instanceof**	**public**	**transient**
case	**false**	**int**	**return**	**true**
catch	**final**	**interface**	**short**	**try**
char	**finally**	**long**	**static**	**void**
class	**float**	**native**	**super**	**while**
const	**for**	**new**	**swicth**	**continue**

Muuttujan nimen ja arvon määrittäminen ei yksin riitä! Tarvitaan tietotyyppi, joka kertoo muuttujan tyypin (kokonaisluku, merkki tai liukuluku) ja tilanvarauksen (muuttujan koko tavuissa). Esimerkiksi edellä olleen muuttujan a tietotyyppi voisi olla kokonaisluku ja tilanvaraus siten 4 tavua. Tällöin se merkitään ohjelmakoodissa:

```
int a = 1 + 1;
```

Jokaisella muuttujalla ja lausekkeella on oltava tietotyyppi. Tietotyyppi määrittelee lukuarvon sallitun alueen (suurimman positiivisen ja pienimmän negatiivisen luvun), jonka muuttuja tai lauseke voi sisältää. Myös eri laskutoimitusten tulosten on mahduttava määrättyyn tyyppiin. Oletuksena kaikilla lausekkeilla on int-tietotyyppi, mikäli kyseinen lukuarvo mahtuu siihen. Ellei mahdu tietotyyppi muutetaan automaattisesti long-tyypiksi.

Java-kielessä tietotyyppejä on kahdeksan. Javan kaikki tietotyypit ovat **etumerkillisiä** ja niiden **varaustila on käyttöjärjestelmästä riippumaton**. Esimerkiksi int-tietotyypin koko on 32 bittiä riippumatta siitä, minkä käyttöjärjestelmän alaisuudessa ohjelmaa suoritetaan. Seuraavassa on esitetty tietotyypit kokonaisluvuille.

Nimi	Tilanvaraus
byte	8 bittiä
short	16 bittiä
int	32 bittiä
long	64 bittiä

Esimerkki:

```
byte tavu = 255;
```

Kokonaisluku voidaan pakottaa long-tyyppiseksi (64-bittiä), jos luvun perässä esiintyy loppuliite L tai l. Muutoin luku on tyyppiä int, jos se mahtuu etumerkillisen 32-bittisen luvun rajoihin.

2L	Luku 2 muutetaan 64-bittiseksi

Hieman erikoinen tietotyyppi on **boolean**. C:ssä boolean-tietotyyppi on keinotekoinen (tehty DEFINE-lauseella), mutta Java-kielessä se ei ole keinotekoinen. Sen vuoksi boolean-muuttujaan ei voi sovittaa lukuja, vain arvot tosi (**true**) tai epätosi (**false**). Boolean-muuttujalla ei voi myöskään suorittaa laskutoimituksia!

Javassa on lisäksi sellainen erikoisuus C:hen nähden, että merkkijonot ovat **aina** olioita. Tosin char-tietotyyppiä käyttäen voidaan tehdä merkkitaulukko, jota voidaan käyttää tekstin tulostuksessa. Char-tietotyypin tilanvaraukseksi on taulukossa laitettu 16-bittiä. Taulukossa ei ole virhettä, vaan Java-kieli käyttää tavallisen ASCII-merkistön sijasta 16-bittistä UNICODEa.

Nimi	Tilanvaraus
boolean	8 bittiä
char	16-bittiä

Esimerkki:

```
boolean lippu = true;
```

```
char merkki = ‘a ‘;
```

Seuraavassa on esitetty liukulukujen tietotyypit (liukuluvut on määritelty tarkasti IEEE 754 -standardissa). Liukuluku koostuu kokonaisluvusta, desimaalipisteestä ja desimaaliosasta sekä eksponentista. Liukuluku voidaan pakottaa ekspotentiaaliseksi, jos luvun keskellä esiintyy kirjain E tai e. Seuraavassa on esimerkkejä liukuluvuista.

Nimi	Tilanvaraus
float	32-bittiä
double	64-bittiä

Esimerkiksi:

```
float f = 3.14;
double d = 3.14E12;      // =3.14 * 10^12
```

Liukuluku voidaan pakottaa double- tai float-tyyppiseksi, jos luvun perässä esiintyvät kirjaimet D,d tai F,f.

2.0D	double
2.0F	float

Muuttujia ei tarvitse esitellä yksitellen usealla rivillä, jos tietotyyppi pysyy samana. Muuttujan arvoakaan ei tarvitse heti antaa esittelyn yhteydessä.

```
float a, b, c;

    a = 1.0;
    b = 2.2;
    c = 128.6;
```

3.5 Tyyppimuunnokset

Tyyppimuunnokset ovat Java-ohjelmoinnissa arkipäiväinen asia. Tyyppimuunnosta tarvitaan aina, kun jonkun muuttujan sisältö kopioidaan toiseen, tietotyypiltään erilaiseen muuttujaan. Tyyppimuunnos on käytännössä sitä, että kopioitavan muuttujan tietotyyppi tulkitaan kohteen tietotyypiksi.

Tyyppimuunnosta ei tietenkään tarvita, jos muuttujat ovat tietotyypiltään samanlaisia. Esimerkiksi tapauksessa, jossa muuttujan A sisältö asetetaan muuttujan B sisällöksi ja kummankin muuttujan tietotyyppi on int.

```
int A,B;

    B = A;
```

Jos B olisikin tietotyypiltään double, käännösvaiheessa siitä tulisi virheilmoitus että muuttuja B ei ole int-tietotyyppiä. Tyyppimuunnos poistaa tämän virheilmoituksen.

```
int A;
double B;

    B = (int) A;
```

Tosin double-tyyppisen muuttujan arvo ei voi kopioitua sellaisenaan int-tietotyyppiin - arvo on doublen esittämä kokonaisluku. Pyöristystä ei tapahdu!

```
public class testi
{
 public static void main(String args[])
 {
  int i;
  double l = 3.69;

  i = (int) l;
  System.out.println("I:n arvo on"+i);
 }
}
```

tulos:

```
I:n arvo on 3
```

Sunin Java-kielen dokumentaatiossa on hyvä taulukko, joka kertoo kuinka paljon tyyppimuunnos ”maksaa”. Jos arvo on suurempi kuin 10, tapahtuu tiedon häviämistä.

From \ To	byte	short	char	int	long	float	double
byte	0	1	2	3	4	6	7
short	10	0	10	1	2	4	5
char	11	10	0	1	2	4	5
int	12	11	11	0	1	5	4
long	12	11	11	10	0	6	5
float	15	14	13	12	11	0	1
double	16	15	14	13	12	10	0

Note: Cost >= 10 causes data loss.

Java-kieli sallii tyyppimuunnokset myös luokkien välillä, eli luokka A voidaan muuttaa luokaksi B. Luokkien ja muuttujien välinen suora tyyppimuunnos ei ole mahdollinen. Vaikka luokan tarkoituksena olisikin esittää jotakin tietotyyppiä, esimerkiksi kokonaislukua (Integer-luokka), niin sen muuttaminen int-tietotyypiksi ei käy tyyppimuunnoksella. Se on tehtävä luokan funktioilla.

3.6 Lukujärjestelmät

Lukujärjestelmiä on käytettävissä kolme: desimaali eli kymmenjärjestelmä, heksadesimaali eli kuusitoistajärjestelmä ja oktaali eli kahdeksanjärjestelmä. Kaksi- eli binäärijärjestelmää ei ole lainkaan käytettävissä: Javassa ei ole muutenkaan bittikentän operaattoreita, vain loogiset operaattorit. Oletuslukujärjestelmä on kymmenjärjestelmä.

Desimaaliluku on puhdas luku (se ei saa alkaa nollalla)

- Kantaluku 10
- Esimerkki: 100

Heksadesimaaliluku alkaa 0x-yhdistelmällä

- Kantaluku 16
- Esimerkki: 0x100

Oktaaliluku alkaa nollalla

- Kantaluku 8
- Esimerkki: 0100

3.7 Merkkijonot

Merkkijono alkaa ja päättyy lainausmerkkeihin ja se saa sisältää mitä tahansa merkkejä. Kenoviivalle on varattu erikoiskäyttö, kuten C:ssä. Kenoviivan avulla voidaan merkkijonoon sisällyttää erikoismerkkejä, kuten rivinvaihto ja lainausmerkit.

Mikäli merkkijonoon halutaan sisällyttää pelkkä kenoviiva, se kirjoitetaan kahteen kertaan. Tämä on huomattava erityisesti kirjoitettaessa tiedostojen nimiä. Kenoviivan avulla voidaan merkkijono jakaa useammalle riville lähdekoodissa siten, että käännettäessä merkkijono on kuitenkin yhtenäinen.

rivi jatkuu	\
rivinvaihto	\n
vertikaalinen tabulointi	\t
askelpalautin BS	\b
palautus rivin alkuun CR	\r
sivun vaihto FF	\f
kenoviiva	\\
heittomerkki	\'
lainausmerkki	\"
oktaalikuku	\ddd
heksaluku	\xdd
merkki (unicode)	\udddd

Esimerkiksi:

```
public class testi
{
 public static void main(String args[])
 {
  System.out.println("Tietotyypit ovat:\n\tInt\n\tByte");
 }
}
```

Tulos:

```
Tietotyypit ovat:
    Int
    Byte
```

Merkkijonot Javassa ovat **olioita**, kuten edellä on jo mainittukin. Tästä ei kuitenkaan kannata pelästyä, vaikka ei olisikaan ennestään tutustunut olioihin. Olion käyttö merkkijonoissa ei vaikeuta asioita vaan päinvastoin helpottaa: määrittelyn yhteydessä ei tarvita taulukkoa. Olio suorittaa tilanvarauksen itse, joten muistin vahingoittumisen vaaraa ei ole.

Merkkijonon muodostetaan String-luokalla. Tulostettaessa annetaan funktiolle, esimerkiksi println-funktiolle, parametriksi String-olio. Funktio osaa automaattisesti tulostaa tekstin oliosta. String-olio sisältää myös useita ohjelmoijalle käteviä funktioita. Niistä kerrotaan tarkemmin kappaleessa *Merkkijonojen käsittely*.

```
String s = "Olen merkkijono";
```

Nyt s on olio, joka sisältää tekstin "Olen merkkijono". Teksti voidaan tulostaa näytölle tekstintulostusfunktioilla, jotka sallivat parametrikseen merkkijonon. Esimerkiksi:

```
public class testi
{
 public static void main(String args[])
 {
  String s = "Olen merkkijono";
  System.out.println(s);
 }
}
```

Tulos:

```
Olen merkkijono
```

3.8 Merkkijonojen käsittely

Merkkijonojen käsittely tapahtuu String-olion sisältämillä funktioilla. Tällöin ei tarvita erillistä ohjelmakirjastoa, jossa olisi merkkijonon käsittelyfunktioita (esim. string.h).

Kun merkkijono luodaan String-luokkaa käyttäen, olion sisällöksi asetetaan jokin merkkijono. Myöhemmin olion sisältö voidaan muuttaa kirjoittamalla siihen uusi merkkijono (tilanvaraus suoritetaan automaattisesti). C-kielessähän merkkitaulukon sisällön vaihtamiseen tarvitaan funktio strcpy. Lisäksi C:ssä pitää tarkkailla koko ajan taulukon kokoa. Java on tässä mielessä paljon joustavampi kuin C, koska taulukon koon varaus on automaattista.

```
public class testi
{
 public static void main(String args[])
 {
  String nimi = "Olen merkkijono";
```

```
  System.out.println(nimi);

  nimi = "Nyt olen toinen merkkijono";
  System.out.println(nimi);
 }
}
```

Tulos:

```
Olen merkkijono
Nyt olen toinen merkkijono
```

Merkkijonoilla on eräs tärkeä ominaisuus. Merkkijonoja voidaan yhdistää plus-operaattorilla aivan kuin lukuja lasketaan yhteen. Kahden merkkijonon yhdistäminen käy siten seuraavasti:

```
s = "Merkkijono"+"Merkkijono";
```

Tulos:

```
MerkkijonoMerkkijono
```

+ -operaattorin avulla voidaan merkkijonoon sisällyttää myös muita tietotyyppejä, esimerkiksi int-tietotyyppejä. Tällöin tietotyypin sisältämä lukuarvo muunnetaan automaattisesti vastaaviksi merkeiksi.

```
int A = 10;

s = "Muuttujan A arvo "+A+" tulostetaan tässä";
```

Tulos:

```
Muuttujan A arvo 10 tulostetaan tässä
```

Koska merkkijono on olio, sen sisältämiä merkkejä ei voi suoraan osoittaa eikä sitä voi tyyppimuunnoksella muuttaa char-taulukoksi. Mutta ei hätää, String-olio sisältää kaikki tarvittavat funktiot itsessään. Esimerkiksi funktiolla **`length`** saadaan luettua merkkijonon pituus.

```
int pituus;
String s;

s = "Olen merkkijono pituudeltani monta merkkiä";
pituus = s.length();
```

```
System.out.println(""+pituus);
```

Tulos:

```
42
```

Olioiden sisältämiä funktioita voidaan osoittaa siis kirjoittamalla **olion nimi, piste ja funktion nimi**. Tällä tavoin saadaan käyttöön monia muitakin String-luokan funktioita: Niistä tärkeimpiä on esitetty seuraavassa.

charAt(int)	Palauttaa merkin, joka on tietyssä indeksissä
compareTo(String)	Vertaa merkkijonoa toiseen merkkijonoon
concat(String)	Liittää uuden merkkijonon tämän merkkijonon loppuun
equals(Object)	Vertaa merkkijonoa toiseen merkkijonoon
length()	Palauttaa merkkijonon pituuden
replace(m,u)	Muuttaa merkkijonon merkit m merkeiksi u
toCharArray()	Muuttaa merkkijonon merkkitaulukoksi
toLowerCase()	Muuttaa merkkijonon pieniksi kirjaimiksi
toUpperCase()	Muuttaa merkkijonon isoiksi kirjaimiksi

Esimerkiksi:

```
public class testi
{
 public static void main(String args[])
 {
  String s1 = "PALO";
  String s2 = "auto";

  System.out.println(s1+s2);

  s1 = s1.toLowerCase();
  s2 = s2.toUpperCase();
```

```
  System.out.println(s1+s2);
 }
}
```

Tulos:

```
PALOauto
paloAUTO
```

String-olio saadaan muutettua tarvittaessa merkkitaulukoksi funktiolla **`toCharArray`**. Toimenpidettä tarvitaan, kun halutaan tulostaa merkkejä yksitellen.

```
String s = ”Merkkijono”;
char m []= new char [10];
    m = s.toCharArray();
```

String-olion funktioita voi käyttää myös siten, että määritellään pelkkä merkkijono ilman vastaavaa muuttujan nimeä. Osoitetaan vain funktioita suoraan merkkijonossa.

```
"merkkijono".compareTo("merkkijono");
```

3.9 Taulukot ja niiden käsittely

Taulukoina Javassa voivat olla kaikki tietotyypit ja oliot. Taulukkomuuttuja luodaan siten, että määritellään tietotyyppi, muuttujan nimi ja taulukko merkeillä []. Taulukon koko varataan aina alkioina, ei esimerkiksi tavuina (alkion koko riippuu tietotyypin koosta). Javassa taulukon kokoa ei kerrota taulukkomuuttujan määrittelyn yhteydessä.

```
int taulukko[];
```

Edellä tapahtui vasta taulukon määrittely, varsinaista muistinvarausta ei ole vielä suoritettu. Muistinvaraus tapahtuu new-operaattorilla.

```
int taulukko[] = new int [20];   // varaa muistia
```

Taulukon alkioita voidaan osoittaa siten, että ensimmäinen alkio on 0, seuraava 1 ja niin edelleen. Mikäli osoitetaan taulukon ulkopuolella olevaa alkiota, generoidaan poikkeustilanne. Myös negatiivisesta indeksistä generoidaan poikkeustilanne. Java siis huolehtii, ettei muistia voi käsitellä huolimattomasti.

```
public class testi
{
 public static void main(String args[])
 {
  int taulukko[];

  taulukko = new int [20];
  taulukko[0] = 20;

  System.out.println("Ensimmäisen alkion sisältö
                                on "+taulukko[0]);
 }
}
```

Tulos:

```
Ensimmäisen alkion sisältö on 20
```

Myös moniulotteisia taulukoita voidaan tehdä Java-kielellä. Se tapahtuu samalla tavalla kuin edellä, erona vain on taulukoiden lukumäärä. Muistin varauksen yhteydessä pitää määritellä vähintään taulukon indeksien lukumäärä (indeksiin kuuluvien alkioiden lukumäärän ei tarvitse olla tiedossa).

```
int taulukko [][] = new int [10][];
```

tai

```
int taulukko [][] = new int [10][10];
```

Olioita voidaan käyttää taulukkoina, yksi- tai moniulotteisina. Taulukoiden alustaminen olioille ei ole sen vaikeampaa kuin tietotyypeillekään. Seuraavassa varataan 10 alkiota merkkijonolle.

```
String  merkkijonot[] = new String[10] ;
```

Taulukot voidaan myös alustaa lukuarvoilla välittömästi taulukon määrittelyn yhteydessä seuraavasti:

Yksiulotteinen:

```
int taulukko[] = {1, 2, 3};
```

Moniulotteinen:

```
int taulukko[][] = { {1, 2, 3},
                     {4, 5, 6},
                     {7, 8, 9 } };
```

3.10 Operaattorit

Operaattoreilla on Java-kielessä suuri merkitys. Operaattorien avulla tehdään laskutoimituksia, verrataan lukuja toisiinsa ja sijoitetaan lukuja muuttujiin. Seuraavana on tarkoitus perehtyä sijoitusoperaattoriin, aritmeettisiin-, vertailu- ja loogisiin operaattoreihin.

Sijoitusoperaattori on tuttu on-merkki `=`. Sitä ei saa sekoittaa vertailuoperaattoriin `==`, koska se ei sijoita lukua vaan vertaa lukujen yhtäsuuruutta.

```
int a = 10; // Sijoitus a = 10;
```

Aritmeettiset operaattorit on tarkoitettu lukujen laskutoimituksiin. Kielessä on valmiina vain peruslaskutoimitukset, mutta esimerkiksi neliöjuuri ja potenssiinkorotus suoritetaan Math-luokan funktioilla.

+	yhteenlasku
-	vähennyslasku
*	kertolasku
/	jakolasku
%	jakojäännös
~	kahden komplementti

```
int a = 10%3     // a = jakolaskun 10/3
                    jakojäännös eli 1
```

Arvonmuunto-operaattorit on tarkoitettu yhden muuttujan arvon muuntamiseen. Muuttujan arvoon on mahdollista lisätä tai vähentää siitä yksi, lisätä/vähentää toinen luku tai tehdä arvosta negaatio (vaihtaa luvun etumerkki). Myös kerto- ja jakolaskutoimitukset toisella luvulla ovat mahdollisia.

++ lisää yksi

-- vähennä yksi

+= lisää luku

-= vähennä luku

*= kerro luku

/= jaa luku

```
a += 256;        // a = a + 256
```

Vertailuoperaattoreita tarvitaan silloin, kun lukuja pitää verrata toisiinsa. Operaattorit ottavat huomioon luvun **etumerkin,** joten negatiivinen luku on aina pienempi kuin positiivinen. Mikäli halutaan verrata lukuja itseisarvojen perusteella, pitää käyttää Math-luokasta löytyvää **abs**-funktiota. Vertailuoperaattoreita käytetään yleensä vertailulauseilla, kuten if-lauseessa.

== yhtäsuuri

!= erisuuruus

> suurempi kuin

< pienempi kuin

<= pienempi tai yhtäsuuri kuin

>= suurempi tai yhtäsuuri kuin

```
esim: if (a > b)    // jos a suurempi kuin b
```

Loogiset operaattorit ovat hyviä silloin, kun tarvitaan ehtoja lausekkeiden välillä.

! EI-operaattori

&& JA-operaattori

|| TAI-operaattori

Esimerkiksi jos muuttujien **a** ja **b** tulee olla arvoiltaan 5 ja 10, käytetään JA-operaatiota seuraavalla tavalla varmistamaan tämä.

```
if (a == 5  &&  b == 10)
```

Bittitason operaattorit toimivat nimensä mukaisesti bittitasolla. Operaattoreita ei käytetä ehtojen asettamiseen, kuten yllä olevia loogisia operaattoreita. Bittitason operaatiot kohdistuvat enemmän luvun bittien nollaamiseen ja asettamiseen.

!	EI -operaattori
&	JA -operaattori
|	TAI -operaattori
^	XOR -operaattori
-	negaatio

Jos bittioperaattorit ovat outoja, seuraavassa on lyhyesti esitetty mitä kullakin loogisella operaatiolla voi tehdä.

- EI-operaatiota käytetään kääntämään bittien arvot
- JA-operaatiota käytetään nollamaan luvun tietyt bitit
- TAI-operaatiota käytetään asettamaan luvun tietyt bitit

Seuraavassa on esitetty operaatioita koskevat totuustaulukot kahden muuttujan avulla. A ja B ovat bitit, joihin operaatio kohdistuu. F on tulos.

OR

A	B	**F**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

AND

A	B	**F**
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

XOR

A	B	**F**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

NOT

A	F
0	1
1	0

Esimerkiksi luku 127 koostuu biteistä seuraavasti:

```
0111 1111
```

Kun luvun kanssa tehdään JA-operaatio luvulla 1, asettuvat vain ne bitit, jotka kummassakin luvussa ovat ykkösiä (vrt. totuustaulukko). Eli tässä tapauksessa lopputulos on:

```
0000 0001
```

Muita bittitason käsittelyyn liittyviä operaattoreita ovat bittien siirto-operaattorit. Ne nimensä mukaisesti siirtävät luvun bittejä vasemmalle tai oikealle. Tällä tavoin voidaan suorittaa kerto- ja jakolasku kahden potensseilla (2,4,8 jne).

<<	Kierrätä vasemmalle
>>	Kierrätä oikealle (kopioi etumerkkiä)
>>>	Kierrätä oikealle (etumerkki positiiviseksi)
<<=	Kierrätä vasemmalle ja sijoita
>>=	Kierrätä oikealle ja sijoita (kopioi etumerkkiä)
>>>=	Kierrätä oikealle ja sijoita (etumerkki positiiviseksi)

```
a = a << 2       // kierrättää muuttujan a:n
                 bittejä vasemmalle kahdella eli
                 tulos kerrotaan neljällä
```

Operaattorien suoritusjärjestys Javassa on seuraava

```
. [] ()
++ -- ! ~ instanceof
* / %
+ -
<;<; >;>; >;>;>;
<; >; <;= >;=
== !=
&;
^
|
&;&;
||
?:
= op=
,
```

3.11 Ohjauslauseet

Ohjauslauseiden avulla ohjataan ohjelmaa. Ne ovat eräänlaisia käskyjä, jotka käskevät tietokonetta tekemään jotakin. Ohjauslauseilla voidaan toteuttaa erilaisia ehtoja ja silmukoita. Seuraavaksi on tarkoituksena tutustua kolmenlaisiin ohjauslauseisiin: ehto-, tapaus- ja toistolauseisiin.

Ehtolause **if - else** koostuu kahdesta haarasta. If-haara testaa onko annettu ehto (=lausekkeen arvo) tosi. Jos ehto on tosi, suoritetaan seuraava lause, muulloin se jätetään suorittamatta.

```
if (ehto)
  lause;
```

Ehtolausekkeeseen voidaan liittää mikä tahansa vertailuoperaattori. Voidaan esimerkiksi verrata, onko a suurempi kuin b.

```
if (a>b)
 a=100;          // sijoita a=100, jos a on suurempi
                    kuin b
```

Ehtolauseeseen voidaan liittää myös else-haara. Mikäli annettu ehto ei toteudu, suoritetaan else-haaran jälkeinen lause.

```
if (ehto)
 lause;
else
 lause;
```

Esimerkiksi seuraavassa testataan, onko muuttuja A tosi vai epätosi, ja asetetaan muuttuja B sen mukaan. Jos A on tosi, niin B on tosi ja päinvastoin.

```
boolean a = true;
boolean b = false;

    if (a == true)
      b = true;
    else
      b = false;
```

Peräkkäisten if-else-haarojen avulla voidaan muodostaa mutkikkaampia vertailuja. Else-haaraan voidaan liittää uusi if-haara eli uusi ehtolause, joka testaa seuraavaa ehtoa.

```
if (a<10)
  a=10;
else if (a>20)
  a=20;
```

Esimerkki:

```
public class testi
{
 public static void main(String args[])
 {
  int a, b;
  int luku;

  a = 10;
  b = 20;

  if (a > b)
   luku = a;
  else
   luku = b;

  System.out.println("Suurempi luku on "+luku);
 }
```

Tulos:

```
Suurempi luku on 20
```

Toisentyyppinen ehtolause saadaan aikaan kysymysmerkillä. Mikäli ehto on tosi, suoritetaan lauseke1 ja mikäli ehto on epätosi, suoritetaan lauseke2.

ehto ? lauseke1 : lauseke2

Esimerkiksi:

```
a = (a==b) ? 1 : 2
```

Jos ehto on tosi eli a on yhtäsuuri kuin b, muuttujaan a sijoitetaan arvo 1. Jos ehto on epätosi, muuttujaan a sijoitetaan arvo 2.

```
public class testi
{
 public static void main(String args[])
 {
  int a, b;
```

```
  a = 10;
  b = 20;

  int luku = (a>b) ? a : b;
  System.out.println("Suurempi luku on "+luku);
 }
}
```

Tulos:

```
Suurempi luku on 20
```

Tapauslause **switch-case** on periaatteiltaan samanlainen kuin if-lause. Se tutkii annettua ehtoa switch-rakenteessa ja haarautuu vastaavaan case-haaraan. Break-lauseella lopetetaan case-haaran suorittaminen, eli breakin käyttö on pakollista, ellei välttämättä haluta jatkaa seuraavasta case-haarasta. Tapauslause on muotoa:

```
switch (muuttuja)
{
 case arvo:
  break;
}
```

Mikäli joku case-haaran lukuarvo vastaa switchin sisältöä, niin se suoritetaan. Jos mikään case-haaran arvo ei vastaa switch-lauseen sisältöä, suoritetaan default- eli oletushaara. Oletushaaran käyttäminen ei ole kuitenkaan pakollista. Seuraavassa esimerkissä testataan switch-haarassa muuttujan a arvoa. Koska se on 101, suoritetaan case-haara 101 ja muuttujan b arvoksi asetetaan epätosi.

```
int a = 101;
boolean b;
switch (a)
{
    case 100:
       b=true;
       break;
    case 101:
       b=false;
       break;
    default:
       break;
}
```

Toistolauseita on olemassa kaksi. Ne ovat nimeltään **for** ja **while**. Ensimmäistä käytetään, kun tiedetään kuinka monta kertaa silmukkaa suoritetaan. Jälkimmäistä käytetään, kun ei tunneta silmukoiden tarkkaa lukumäärää.

For-toistolause muodostuu kolmesta osasta. Ensin alustetaan laskuri, sen jälkeen on toistoehto ja kolmanneksi lisätään laskuria (yleensä yhdellä). For-lauseen muoto on seuraava:

```
for (alkuarvo ; ehto ; lisäys)
  lause;
```

For-rakenteen jälkeistä lausetta (tai lauseita, jos ne on yhdistetty aaltosuluilla) suoritetaan niin monta kertaa kuin ehto on voimassa.

Esimerkki:

```
public class testi
{
 public static void main(String args[])
 {
  int i;
  for (i=0; i<5; i++)
   System.out.println("I:n arvo on"+i);
 }
}
```

Tulos:

```
I:n arvo on 0
I:n arvo on 1
I:n arvo on 2
I:n arvo on 3
I:n arvo on 4
```

Toistoehdon for-lauseessa ei tarvitse olla yhdistetty laskuriin, vaan se voi olla mikä tahansa suoritusehto. Esimerkiksi seuraavassa silmukka suoritetaan niin monta kertaa kunnes summa on 499000.

```
public class testi
{
 public static void main(String args[])
 {
  int i;
  int summa = 0;

  for (i=1; summa<=499000; i++)
```

```
   summa+=i;

  System.out.println("I:n arvo on "+i);
 }
}
```

tulos:

```
I:n arvo on 1000
```

Muuttujan i määrittely voi tapahtua myös for-sanan sisällä.

```
for (int i = 0; i < 10; i++)
 lause;
```

While-toistolause muodostuu pelkästä ehdosta ja sitä seuraavasta lauseesta. While-lausetta toistetaan niin kauan kuin ehto on tosi. Jos ehto on epätosi jo while-rakenteeseen mennessä, sitä ei suoriteta kertaakaan. While-toistolause on muotoa:

```
while (ehto)
 lause;
```

Esimerkiksi seuraavassa silmukkaa suoritetaan niin kauan kunnes i on suurempi kuin 10. Käytännössä 4 kertaa.

```
int i=0;

while (i<10)
  i += i;
```

Do-haara voidaan liittää while-haaraan, jolloin syntyy **do-while-toistolause**. Do-while-lause suoritetaan aina vähintään yhden kerran, vaikka ehto olisi epätosi. Do esiintyy lähdekoodissa ennen while-rakennetta sisältäen varsinaisen ohjelmakoodin. While testaa vain ehdon. Do-while-toistolause on muotoa:

```
do
  lause;
while (ehto);
```

```
public class testi
```

```
{
 public static void main(String args[])
 {
  int summa = 0;

  do
  {
   summa++;
   System.out.println("Silmukka suoritettu "+summa+"
                                                kertaa");
  }
  while (summa == 1);
 }
}
```

Tulos:

```
    Silmukka suoritettu 1 kertaa;
```

Java-kieli tukee myös silmukoita, joille annetaan jokin nimi. Nimetty silmukka alkaa silmukan nimellä ja päättyy kaksoispisteeseen. Vain seuraava lause kuuluu silmukkaan!

```
    nimi:
     lause;
```

Nimetty silmukka sisältää esimerkiksi for-lauseen, joka muodostaa suoritettavan silmukan. Nimetyssä silmukassa voi olla ehdottomia hyppykäskyjä, kuten break ja continue, joiden avulla silmukan suorittaminen voidaan pysäyttää tai aloittaa alusta. Hyppykäskyjen perään liitetään silmukan nimi. Katso seuraava esimerkki:

```
public class testi
{
 public static void main(String args[])
 {
  int i = 0;

  silmukka:
   for (i=0; i<1000; i++)
    if (i > 100)
     break silmukka;

  System.out.println("Silmukka suoritettu "+i+" kertaa");
 }
```

```
}
```

Tulos:

```
Silmukka suoritettu 101 kertaa
```

Yllä olevassa ohjelmakoodissa on käytetty **break**-lausetta, joka esiintyi jo tapauslauseessa. Break-lauseen tehtävänä on keskeyttää ohjauslauseen tai jonkun haaran suorittaminen.

Toinen tärkeä hyppylause on **continue**. Continue-lauseella pysäytetään ja aloitetaan alusta ohjauslauseen suorittaminen. Vaikka luvun alussa esitetyssä varatut sanat -taulukossa oli esillä goto-hyppylause, se ei ole kuitenkaan toiminnassa.

Olio-ohjelmoinnin yhteydessä tutustumme jälleen uusiin lauseisiin. Tässä vaiheessa voidaan vielä esitellä **return**-lause, jota käytetään aliohjelmakutsusta palautuessa. Return-lauseen avulla palautetaan jokin arvo jäsenfunkiota kutsuneelle ohjelmalle.

```
int jasenfunktio()
{
  return 2;
}
```

Aliohjelmia eli funktioita pääsemme tekemään vasta olio-ohjelmoinnin puolella.

3.12 Olio-ohjelmointi

Olio-ohjelmointiin liittyy useita tärkeitä termejä, joista jo muutamia on tässä luvussa mainittukin. C++-kielen olio-ohjelmointi ei ole monellekaan niin tuttua eikä helppoa kuin puhdas C-kielinen ohjelmointi. Nyt onkin tarkoituksena perehtyä olio-ohjelmoinnin saloihin. Java-kielen olio-ohjelmointi poikkeaa kieliopiltaan C++-kielen olio-ohjelmoinnista, joten myös ne, joille aihe on ennestään tuttua, joutuvat tutustumaan Javan kielioppiin.

Aivan aluksi tutustutaan hieman olio-ohjelmoinnin termeihin. Olio-ohjelmoinnin tärkein käsite on **luokka**. Luokka voi sisältää funktioita (=jäsenfunktioi eli metodi) ja muuttujia. Puhutaan **kapseloinnista** (*encapsulation*), kun samassa luokassa on funktioita ja muuttujia. Luokan muuttujat ja funktiot voidaan asettaa salaisiksi, sillä tarkoitetaan silloin **datan kätkentää** (*data hiding*). Myös sanat **staattinen** ja **dynaaminen** ovat paljon esillä olio-ohjelmoinnissa: staattinen merkitsee pysyvää ja dynaaminen sen vastakohtaa. Sanat ovat esillä kun puhutaan muuttujien vaikutusalueesta. Luvun aikana tullaan tutustumaan tarkemmin muun muassa seuraaviin asioihin.

- luokka
- jäsenmuuttuja
- funktio
- periytyvyys
- ylikuormitus
- ylikirjoitus
- muodostin
- hajotin
- this-osoitin
- super-osoitin

Javan ohjelmointirajapinta (API) muodostuu täysin luokista. API sisältää esimerkiksi luokan Label, joka näyttää suunnilleen tältä:

Variable index

CENTER
LEFT
RIGHT

Constructor index

Label()
Label(String)
Label(String, int)

Method index

addNotify()
getAlignment()
getText()

paramString()
setAlignment(int)
setText(String)

Kaikki muutkin API:n luokat ovat samantyylisiä. Ne koostuvat muuttujista (*variable index*), muodostimista (*constructor index*) ja funktioista (*method index*). Luvuissa 4 - 12 tutustutaan useisiin API:n luokkiin.

3.13 Luokka

Luokka esittää tyypillistä olio-ohjelmoinnin mallia. Kun puhutaan oliosta, sillä tarkoitetaan käyttöönotettua luokkaa. Toisin sanoen olio on luokan esiintymä. Aivan samoin kuin A voi olla int-tietotyypin muuttuja, nyt tietotyypin tilalla on vain luokka. Itse luokka muodostuu struktuurin (C-kielessä) tapaisesta rakenteesta, jossa on sekä muuttujia että funktioita.

Luokan määrittely tehdään lauseella class. Luokalle voidaan antaa (ei ole välttämätön) muunnin (*modifiers*), jonka tehtävä on määritellä luokan tyyppi samoin kuin tietotyyppi määrittelee muuttujan tyypin. Lisäksi voidaan määrätä mistä luokka on periytynyt extends-sanalla, ja lopuksi annetaan mahdollinen rajapinta implements-sanalla.

```
muunnin class luokka extends kantaluokka
                     implements rajapinta
```

Muuntimet ovat luokkien, funktioiden ja jäsenmuuttujien osoittamiseen vaikuttavia tekijöitä. Tässä yhteydessä käsitellään vain luokkiin kohdistuvia muuntimia, myöhemmin kerrotaan muuttujille ja funktioille sopivista muuntimista. Luokan yhteydessä muuntimella määrätään, mitkä ovat kyseisen luokan ominaisuudet: voiko se periytyä, onko se yleinen jne. Jokainen luokka asetetaan muuntimella yksityiskohtaisesti. Alla olevassa taulukossa on esitetty luokalle sopivia muuntimia. Periytyvyyteen ja rajapintoihin perehdytään hieman myöhemmin.

Muunnin	Selitys
public	Luokka periytyy ja sen sisältämiä funktioita ja muuttujia voi osoittaa mistä luokasta tahansa.
final	Luokka ei periydy. Luokan sisältöä ei voi muuttaa. Funktioita ja jäsenmuuttujia voi osoittaa mistä luokasta tahansa.
abstract	Luokka on periytettävä! Luokasta ei voi tehdä oliota muulloin kuin periyttämällä. Luokan funktiot eivät sisällä ohjelmakoodia.

Luokka on jo tullut tutuksi tämän luvun aikana esimerkkiohjelmien yhteydessä. Jokainen tehty esimerkkiohjelma edustaa luokkaa (luokan nimenä on ollut testi). Luokkaan määritelty muuttujia (esim. int i) ja funktio main.

```
public class testi
{
 public static void main(String args[])
 {
  int i = 0;
   for (i=0; i<1000; i++)
    if (i > 100)
     break;

  System.out.println("Silmukka suoritettu "+i+"
                                        kertaa");
 }
}
```

Luokan ei tarvitse aina olla Java-sovellus tai appletti, se voi olla esimerkiksi mikä tahansa uusi tietotyyppi. Ohjelmia tehtäessä voidaan tehdä tietotyyppi, joka kuvaa todellista tehtävää, esimerkiksi yrityksen henkilötietorekisteriä. Henkilötiedot koostuvat asiakkaan nimestä, osoitteesta ja puhelinnumerosta.

```
String etunimi;
String sukunimi;
String osoite;
String puhelinnumero;
```

Lisäksi tarvitaan funktiot, joilla tietoja voidaan muuttaa, poistaa ja lisätä rekisteriin. Funktiot ovat:

```
void MuutaSukunimi (String uusinimi);
void MuutaOsoite(String uusiosoite);
void MuutaPuhelinnumero(String uusinumero);
```

Ilman luokkaa näiden tietojen käsittely olisi erittäin vaikeaa. Luokan avulla muuttujat ja funktiot voidaan kapseloida yhteen kokoelmaan, olioon. Tietojen käsittely on sen jälkeen helppoa.

```
class Henkilo
{
 String etunimi;
 String sukunimi;
 String osoite;
 String puhelinnumero;

 void MuutaSukunimi (String uusinimi)
 void MuutaOsoite(String uusiosoite)
 void MuutaPuhelinnumero(String uusinumero)
}
```

Luokka on siinä mielessä näppärä, koska se kapseloi muuttujat ja funktiot yhteen, ja kätkee luokan tiedot. Loppukäyttäjälle ei näy kuinka luokan funktiot on toteutettu, tai kuinka ne toimivat. Riittää, kun tietää, mikä on luokan käyttöliittymä, eli ne funktiot, joita kutsutaan. Kun edellä esiteltiin API:sta luokka Label, niin luokan käyttäjällä ei tarvitse olla tietoa miten luokka on rakennettu. Pääasia on vain tietää Label-luokan funktiot.

Luokan funktioita ei voi sellaisenaan kutsua eikä luokkaan voi sijoittaa tietoa. Ensiksi pitää tehdä luokan muuttuja eli olio. Se tehdään samaa kaavaa käyttäen kuin muuttuja tehtiin tietotyypillä.

```
int a;
Henkilo henkilotiedot;
```

3.14 Luokan muuttujat

Luokka voi sisältää muuttujia (*instance variables*), joita luokan sisäiset funktiot käyttävät tiedon taltiointiin (muuttujaksi on tulkittava myös olio). Muuttujia kutsutaan myös jäsenmuuttujiksi. Muuttujat ovat kiinteä osa luokkaa, samoin kuin funktiot.

Muuttujat ovat luokan ulkopuolelle näkymättömiä, jos ne määritellään muuntimella yksityiseksi (*private*). Muuttujat näkyvät luokan ulkopuolelle, jos ne määritellään yleisiksi (*public*). Javassa ei

ole C++-kielen tavoin yksityisiä ja yleisiä osia, vaan jokainen muuttuja määritellään yksityiskohtaisesti muuntimella. Esimerkki:

```
class luokka
{
 public String s;   // jäsenmuuttuja s
 private int luku;  // jäsenmuuttuja luku
}
```

Muuttujien määritteleminen luokkaan tapahtuu samalla tavalla kuin muuttujien määritteleminen yleensäkin. Hyvä tapa on määritellä luokan alussa kaikki muuttujat ennen funktioita. Muuttujat ovat käytettävissä luokan sisällä kaikissa funktioissa ja muissa muuttujissa muuntimesta riippumatta.

Funktioiden sisäiset muuttujat, jotka on määritelty funktioiden sisällä, ovat paikallisia muuttujia (*local variables*), ja siten näkymättömiä muille funktioille. Esimerkiksi:

```
public class testi
{
 public static void main(String args[])
 {
  int i = 0;    // paikallinen muuttuja i
 }
}
```

Muunnin	**Selitys**
public	Muuttujaa voi osoittaa mistä luokastaa tahansa.
private	Muuttujaa voi osoittaa vain luokasta, johon se on määritelty. Ei periytyvyyttä!
protected	Muuttujaa voi osoittaa vain alaluokasta.
final	Muuttujaa voi osoittaa mistä luokasta tahansa, mutta sen arvoa ei voi muuttaa.
threadsafe	Muuttujan arvo ei vaihdu, jos kaksi prosessia käyttää sitä samaan aikaan.

3.15 Luokan funktiot

Funktiot ovat ohjelmakoodia, joilla voidaan suorittaa joku toimenpide, esimerkiksi tulostaa tekstiä. Funktiot ovat kiinteä osa luokkaa samoin kuin muuttujat. Myös funktiot voivat hyödyntää muunninta.

Funktioiden määrittely luokkaan tapahtuu seuraavasti: siinä on esiinnyttävä palautusarvo, funktion nimi ja syöttöparametrit. Palautusarvo ja syöttöparametrit voivat olla void-tyyppisiä. Javassa funktiot eivät tarvitse prototyyppiä.

```
Paluuarvo funktion_nimi (parametrit)
```

Funktion ohjelmakoodi kirjoitetaan välittömästi funktion esittelyn yhteyteen aaltosulkujen väliin. Syöttöparametrit, jotka määritellään funktioiden parametrilistassa, eivät voi kätkeytyä funktion sisällä määritetyiltä paikallisilta muuttujilta. Mutta jäsenmuuttujat kätkeytyvät paikallisilta muuttujilta.

```
class luokka
{
 String s;              // kätkeytyy funktion s-
                        // muuttujalta
 int luku;

 void funktio(int s)    // ei kätkeydy
 {
  int s;                // kaksi s-muuttujaa ->
                        // seuraa virheilmoitus

  System.out.println(”Moi!");
 }
}
```

Javassa on myös mahdollisuus määritellä funktioita, joilla ei ole lainkaan ohjelmakoodia. Tällöin funktion määrittely loppuu puolipisteeseen.

```
Paluuarvo funktion_nimi (parametrit);
```

Vartalottomat funktiot määritellään muuntimen avulla. Sellaisia ovat muun muassa **native** ja **abstract**. Native-muuntimella määritellään funktiot, jotka tuodaan jostakin toisesta kielestä, esimerkiksi C:stä.

```
native long timeOfDay();
```

Abstract-tyyppisten funktioiden periaate on määritellä tietynlainen menettelytapa alaluokan funktiolle. Käytettävissä on tietyn nimiset funktiot, joiden tehtävänä on suorittaa jokin tehtävä. Tapa, kuinka tehtävä suoritetaan, jää ohjelmoijan vastuulle. Käytännössä joudutaan siis itse kirjoittamaan rutiinit. Funktio ei voi yksinään olla tyypiltään abstract, vaan koko luokasta tulee abstract, jos yksikin funktio on sellainen.

Muunnin	**Selitys**
public	Funktiota voi osoittaa mistä luokasta tahansa. Funktion ylikuormitus on sallittu, samoin ylikirjoitus, jos ylikirjoittava funktio on myös public.
private	Funktiota voi osoittaa vain luokasta, johon se on määritelty. Ylikuormitus ja ylikirjoitus ovat sallittuja. Funktio ei periydy.
protected	Funktiota voi osoittaa vain alaluokasta. Ylikuormitus on sallittu, samoin ylikirjoitus, jos ylikirjoittava funktio on public tai protected.
final	Funktiota voi osoittaa mistä luokasta tahansa. Funktio ei periydy.
native	Funktiot, jotka tuodaan jostakin muusta kielestä, esimerkiksi C:stä. Funktiosta tehdään prototyyppi Javassa. Ylikuormitus ja ylikirjoitus ovat mahdollisia.
abstract	Funktion käyttö onnistuu vain alaluokasta. Funktioiden vartalo ohjelmoidaan alaluokassa.
synchronized	Funktiota ei voi suorittaa samanaikaisesti useampi säie.

3.16 Luokan käyttöönotto

Luokan käyttöönotto tarkoittaa olion luomista eli luokan muuttujan luomista. Tähän mennessä on vain kirjoitettu luokkia tekemättä niistä olioita. Olion määritteleminen voidaan tehdä näin:

```
luokka olio;              // vain määrittely
```

Pelkkä määritteleminen ei varaa muistia oliolle, joten sen käyttäminenkään ei ole mahdollista. Muistin varaaminen tapahtuu new-operaattorilla. New-operaattori alustaa olion ja kutsuu luokan muodostinta (jokaisella oliolla on parametriton muodostin).

```
olio = new luokka();    // varaa muistia
```

Olion määrittely voidaan tehdä myös samanaikaisesti muistin varauksen kanssa.

```
luokka olio = new luokka();
```

Kun olio on muodostettu, siinä olevia funktioita voidaan kutsua. Funktiokutsun voi suorittaa oliosta itsestään tai olion ulkopuolelta, jos funktio on tyypiltään public. Seuraavassa on esitetty nämä kaksi eri tapaa.

```
funktio(1);          // kutsu olion sisältä
olio.funktio(1);     // kutsu olion ulkopuolelta
```

Kehitetään henkilo-luokkaa hieman eteenpäin. Määritellään luokan funktiot yksityisiksi ja kirjoitetaan funktioille ohjelmakoodi. Lisäksi kirjoitetaan uusi funktio, jolla voidaan tulostaa olion sisältämät tiedot. Tulostuksessa käytetään muotoilumerkkejä \n ja \t.

```
class Henkilo
{
  private String etunimi;
  private String sukunimi;
  private String osoite;
  private String numero;

  public void MuutaSukunimi (String uusinimi)
  {
   sukunimi = uusinimi;
  }
  public void MuutaOsoite(String uusiosoite)
  {
   osoite = uusiosoite;
  }
  public void MuutaNumero(String uusinumero)
  {
   numero = uusinumero;
  }
  public void Tulosta()
  {
   System.out.println("Etunimi: \t"+etunimi);
   System.out.println("Sukunimi: \t"+sukunimi);
```

```
    System.out.println("Osoite: \t"+osoite);
    System.out.println("Puhelin: \t"+numero);
   }
}
```

Luokan lisäksi kirjoitetaan Java-sovellus, joka käyttää luokkaa. Olion alustaminen suoritetaan new-operaattorilla. Olion tiedot täytetään Muuta-funktioilla ja lopuksi tulostetaan kaikki tiedot Tulosta-funktiolla.

```
public class testi
{
 public static void main(String args[])
 {
  Henkilo henkilot = new Henkilo();
  henkilot.MuutaSukunimi("Meikalainen");
  henkilot.MuutaOsoite("Napakuja 2 00000 HKI");
  henkilot.MuutaNumero("12345678");
  henkilot.Tulosta();
 }
}
```

Tulos:

```
Etunimi:    null
Sukunimi:   Meikalainen
Osoite:     Napakuja 2 00000 HKI
Puhelin:    12345678
```

3.17 Muodostimet

Muodostimet ovat myös eräänlaisia funktioita, joita kutsutaan automaattisesti olion luomisen yhteydessä. Myös muodostimet voivat hyödyntää muunninta. Muodostimessa oliolle voidaan asettaa aloitusarvot, kutsua luokan sisäisiä (tai ulkopuolisia) funktioita ja niin edelleen. Muodostimen määrittely tapahtuu seuraavasti:

```
luokan_nimi (parametrit)
```

Muodostimen nimi on sama kuin luokankin. Muodostimella ei ole palautusparametria, vain syöttöparametrit. Muodostimia voi olla useita, jos niillä on erilaiset syöttöparametrit (puhutaan ylikuormittamisesta). Muodostimen ohjelmakoodi kirjoitetaan välittömästi esittelyn yhteyteen aaltosulkujen väliin:

```
class luokka
{
  int s;                // jäsenmuuttuja

  public luokka(int s1) // Parametri s1
  {
   s = s1;              // aseta s = s1
  }
  void funktio(int s)   // funktio
  {
   System.out.println(”Moi!");
  }
}
```

Luokan muodostin voi itse kutsua saman luokan muita muodostimia, jos sellaisia on olemassa. Mikäli näin menetellään, pitää toisen muodostimen kutsun tapahtua heti ensimmäisenä. Tästä kerrotaan lisää this-muuttujan yhteydessä. Muodostimessa voidaan myös kutsua kantaluokan muodostimia. Tästä kerrotaan myöhemmin super-muuttujan yhteydessä.

Olio-ohjelmoinnin alettua tässä luvussa mainittiin hajotin. Hajotin on funktio, joka poistaa olion kuluttaman muistitilan. Se ei ole Java-kielessä käytössä, koska Javassa on automaattinen muistinvapautusmekanismi.

Edellä esitelty Henkilo-luokka tarvitsee myös muodostimen. Kuten huomattiin etunimen muuttamiseen ei ollut funktiota. Sen vuoksi palautui tulos:

```
Etunimi:    null
Sukunimi:   Meikalainen
Osoite:     Napakuja 2 00000 HKI
Puhelin:    12345678
```

Muodostimen avulla voidaan kaikki tiedot antaa yhdellä kertaa. Tällöin funktioille jää nimensä mukainen tehtävä - muuttaa tietoja.

```
class Henkilo
{
  private String etunimi;
  private String sukunimi;
  private String osoite;
  private String numero;
```

```
  public Henkilo(String enimi, String snimi,
                      String os, String puh)
  {
   etunimi = enimi;
   sukunimi = snimi;
   osoite = os;
   numero = puh;
  }

  public void MuutaSukunimi (String uusinimi)
  {
   sukunimi = uusinimi;
  }
  public void MuutaOsoite(String uusiosoite)
  {
   osoite = uusiosoite;
  }
  public void MuutaNumero(String uusinumero)
  {
   numero = uusinumero;
  }
  public void Tulosta()
  {
   System.out.println("Etunimi: \t"+etunimi);
   System.out.println("Sukunimi: \t"+sukunimi);
   System.out.println("Osoite: \t"+osoite);
   System.out.println("Puhelin: \t"+numero);
  }
}
```

Nyt Henkilo-luokka voidaan alustaa seuraavasti:

```
public class testi
{
 public static void main(String args[])
 {
  Henkilo henkilot = new Henkilo(”Matti”,
                     "Meikalainen","Napakuja 2
                     00000 HKI", "12345678");
  henkilot.Tulosta();
 }
}
```

Tulos:

```
Etunimi:    Matti
Sukunimi:   Meikalainen
Osoite:     Napakuja 2 00000 HKI
Puhelin:    12345678
```

3.18 Periytyvyys

Periytyvyys on olio-ohjelmoinnin yksi tärkein ominaisuus. Ilman sitä olio-ohjelmoinnin merkitys ei olisi kovinkaan suuri. Peritymisen ansiosta ohjelmoija välttyy muun muassa toimivan ja testatun ohjelmakoodin uudelleen kirjoittamiselta.

Mitä on periytyminen? Otetaan esimerkki todellisesta elämästä: BMW on auto ja auto on kulkuneuvo. Kulkuneuvon ominaisuudet määrittelevät osaltaan mitkä ovat BMW:n yleiset ominaisuudet. Niin myös Javassa: uusi luokka omistaa alkuperäisen luokan jäsenmuuttujat ja funktiot. Tietenkin BMW:ssä on myös sellaisia ominaisuuksia joita jokaisessa kulkuneuvossa ei ole. Samoin uuteen luokkaan voidaan määritellä funktioita, joita ei ole kantaluokassa.

Luokka, joka periytyy toisesta luokasta, määritellään myös class-sanalla, mutta luokan nimen jälkeen lisätään varattu sana **extends** ja sen luokan nimi, josta periytyminen tapahtuu.

```
class luokan_nimi extends kantaluokka
```

Uusi luokka on nimeltään alaluokka ja alkuperäinen luokka on nimeltään kantaluokka (*superclass*). Alkuperäisen luokan funktioita voidaan ylikuormittaa ja tarvittaessa myös ylikirjoittaa.

```
class Luokka
{
  protected String s;
  public void funktio(String s1)
  {
   s = s1;
  }
}
```

```
class UusiLuokka extends Luokka
{
}
```

Vaikka uuteen luokkaan ei ole määritelty funktioita eikä muuttujia, voidaan kutsua kantaluokan funktioita:

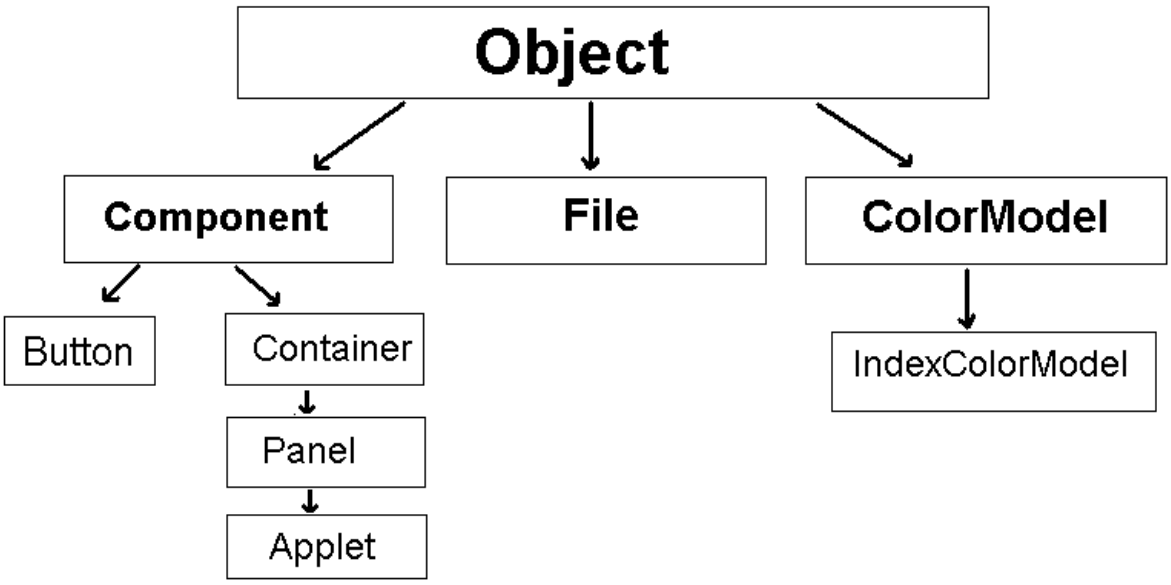
```
UusiLuokka uusiluokka = new UusiLuokka();
uusiluokka.funktio(”Nimi);
```

Periytymisen kannalta muuntimet public, private, ja protected ovat hyvin tarpeellisia. Kun käytetään public-muunninta, luokan kaikki funktiot ja muuttujat näkyvät alaluokalle sekä muille olioille. Kun käytetään private-muunninta, luokan kaikki funktiot ja muuttujat näkyvät vain funktioille. Siksi **protected**-muuntimen käyttö periytymisessä on suotavaa: se estää funktioiden ja muuttujien näkymisen ulospäin muille olioille, mutta sallii niiden näkyä alaluokille. Esimerkiksi:

```
class Luokka
{
  protected String s;
  public void funktio(String s1)
  {
   s = s1;
  }
}
```

Nyt merkkijono s näkyy vain alaluokalle, joka tässä tapauksessa on uusiluokka.

Javassa jokainen luokka on **Object**-luokan alaluokka. Myös kaikki luokat, jotka on tähän mennessä tehty (vaikka niitä ei ole erikseen määritelty periytyväksi). Seuraava kuva havainnollistaa luokkien periytymistä Javan API:ssa.



Lauseella **instanceof** voidaan tutkia mistä luokka on periytynyt. Tämä lause on Javassa varsin käyttökelpoinen, koska kaikki tapahtumat (käyttäjän tekemät toimenpiteet) ovat olioita.

```
if (e instanceof Button)
  // jos e on Button-luokan olio, niin käyttäjä on
     painanut painonappia
```

3.19 Tyyppimuunnokset olioilla

Kieli tukee tyyppimuunnoksia luokkien välillä samoin kuin tietotyyppien välillä. Luokkien välisessä tyyppimuunnoksessa idea on sama: olio tulkitaan toiseksi olioksi. Tyyppimuunnokset rajoittuvat periytyneihin luokkiin: vain alaluokka voi suorittaa tyyppimuunnoksen. Ajon aikainen poikkeustilanne generoidaan, jos tyyppimuunnos on laiton.

```
class luokkaA
{
  String nimi = "Luokka A";
}

class luokkaB extends luokkaA
{
  String nimi = "Luokka B";
}
```

Tyyppimuunnos A:n ja B:n välillä tapahtuu seuraavasti:

```
luokkaA  a;
luokkaB  b = new luokkaB();

a = (luokkaA) b;

System.out.println(a.nimi); // Tulosta sana Luokka A
System.out.println(b.nimi); // Tulosta sana Luokka B
```

3.20 Ylikuormittaminen

Funktion ylikuormittaminen on toiminto, jolla voidaan kirjoittaa useita samannimisiä funktioita. Ylikuormittamisessa funktiolla on sama nimi, mutta erityyppiset palautusparametrit tai syöttöparametrit. Syöttöparametrien määrä saa tietenkin vaihdella.

Seuraavassa on esimerkki ylikuormittamisesta. Siinä funktio ylikuormitetaan kahdesti. Ensimmäinen funktio on parametriton ja asettaa muuttujan sisällöksi luvun 10. Toisella funktiolla on yksi syöttöparametri, joka asettaa muuttujan sisällöksi syöttöparametrin arvon.

```
class luokka
{
  int luku;
  void funktio()
  {
   luku = 10;
  }
  void funktio (int a)
  {
   luku = a;
  }
}
```

Java-kääntäjä valitsee funktion syöttöparametrien mukaan, ei tulosparametrin eikä järjestyksen mukaan. Funktiokutsun yhteydessä valitaan se funktio, jonka syöttöparametrit vastaavat tarkalleen tai eniten määriteltyä funktiota. Mikäli syöttöparametrit vastaavat täysin määritetyn funktion parametreja, tiedon häviötä ei synny (tyyppimuunnosta ei suoriteta). Mutta mikäli syöttöparametrit eivät vastaa funktion parametreja, saattaa tiedon häviämistä tapahtua. Tällöin kääntäjä ilmoittaa siitä.

3.21 Ylikirjoittaminen

Funktion ylikirjoittaminen on tapa, jolla voidaan kirjoittaa uudelleen kantaluokassa esiintyvä funktio. Ylikirjoittamisessa funktiolla on sama nimi, sama palautusparametri ja samat syöttöparametrit. Vain funktion ohjelmakoodi on erilainen. Jos joku näistä tiedoista eroaa, tapahtuu funktion ylikuormittaminen eikä ylikirjoituttaminen. Huomaa, että kääntäjä ei tiedä, kumpaa ohjelmoija on yrittämässä, joten virheilmoitusta ei voi syntyä.

Funktiokutsun yhteydessä valitaan se funktio, joka on kirjoitettu viimeiseksi. Ylikirjoittamisesta on hyötyä esimerkiksi silloin, kun edellisen luokan funktio ei tarjoa kaikkia tarpeellisia ominaisuuksia tai funktio on tyypiltään takaisinkutsuttava. Takaisinkutsuttavia funktioita on Java-ympäristössä runsaasti, koska niiden avulla informoidaan ohjelmaa käyttäjän toiminnoista.

Seuraava ohjelmakoodi periyttää luokan, joka esiteltiin ylikuormittamisen yhteydessä ja ylikuormittaa syöttöparametrittoman funktion.

```
class uusiluokka extends luokka
{
  void funktio()
  {
   luku = 20;
  }
}
```

3.22 This-muuttuja

This-muuttuja esittää nykyistä oliota. Ohjelmakoodissa ei viitata olioon itseensä luokan nimellä vaan this-muuttujalla. This-muuttujaa tarvitaan esimerkiksi silloin, kun olion pitää syöttää itsensä argumenttina toiselle oliolle.

```
class luokka
{
  void omametodi(void)
  {
   new jokuluokka(this);// Antaa itsensä
                        // argumenttina
  }
}
```

This-muuttujan avulla funktioissa voidaan osoittaa jäsenmuuttujia vaikka funktio määrittelisi samannimisiä paikallisia muuttujia (tällöin muuttujat ovat kätkeytyneet).

```
class luokka
{
  int autot;

  void omametodi(int autot)
   {
    this.autot = autot; // asettaa jäsenmuuttujan
                        // autot arvoon 10;
   }
}
```

Kolmas käyttötarkoitus this-muutujalle on luokan toisen (jos olemassa) muodostimen kutsuminen. Tällöin this-muuttujaa käytetään funktiokutsuna.

```
class luokka
{
  int luku;
  public luokka()
  {
   this(10);
  }
  public funktio (int a)
  {
   luku = a;
  }
}
```

3.23 Super-muuttuja

Super-muuttuja on samankaltainen kuin this-muuttuja, mutta super-muuttuja sisältää viittauksen ylempään olioon eli kantaolioon. Super-muuttuja toimii samalla tavoin kuin this, joten sen avulla voi kutsua esimerkiksi kantaluokan funktioita.

```
super.funktio(10);
```

Super-muuttujan käyttökelpoisin tapa on kuitenkin kutsua ylemmän luokan muodostimia.

```
class uusiluokka extends luokka
{
 void omametodi(void)
 {
  super();          // kutsuu kantaluokan
                    // parametritonta muodostinta
 }
}
```

3.24 Staattiset muuttujat ja funktiot

Staattiset muuttujat ja funktiot ovat hieman erilaisia kuin muun tyyppiset muuttujat ja funktiot. Ne ovat enemmän luokkakohtaisia kuin oliokohtaisia. Niitä voidaan osoittaa sekä luokan että olion avulla.

Staattiset muuttujat asetetaan **ensimmäisenä** määriteltyihin arvoihin oliota luotaessa. Sillä ei ole merkitystä, missä järjestyksessä muut kuin staattiset muuttujat määritellään lähdekoodissa, mutta staattiset muuttujat asetetaan annettuihin arvoihin aina esiintymisjärjestyksessä! Esimerkiksi jos muuttuja H on staattinen, sen arvoa voidaan hyödyntää muuttujien määrittelyssä.

```
int y = H + 1;      // Y = 11
static int H = 10;  // H = 10
```

Jos ennen staattisen H-muuttujan määrittelyä on määritelty muita stattisia muuttujia ja yritetään käyttää H:n arvoa hyväksi, syntyy virheilmoitus!

```
static int a = H;   // Virhe! H:ta ei ole vielä
                    // määritelty
static int H = 10;  // H = 10
```

Luokassa ja sen alaluokissa ei voi olla kuin yksi samanniminen staattinen muuttuja tai funktio. Riippumatta siitä, mistä oliosta tai luokasta luetaan staattisen muuttujan arvo, tulos on aina sama. Esimerkiksi:

```
int a = luokka.H;   // H:ta osoitetaan luokan nimellä
int b = H;          // H:ta osoitetaan olion sisällä
```

Muuttujien ja funktioiden lisäksi voidaan määritellä staattinen koodilohko. Se tehdään aaltosulkeilla ja määritellään lauseella static. Koodilohkoa voidaan käyttää esimerkiksi alustamaan luokan muuttujia.

```
static int H;       // staattinen muuttuja

static              // staattinen koodilohko
{
  luokka.H = 10;
}
```

3.25 Rajapinnat

Rajapinta (*interface*) on eräänlainen luokka, jossa määritellään rajapinnan sisältämät muuttujat ja funktiot. Rajapinnan funktioilla ei ole lainkaan ohjelmakoodia. Luokka, johon rajapinta määritellään, saa siis uusia funktioita, mutta ei ohjelmakoodia. Funktion ohjelmakoodi ylikirjoitetaan myöhemmin luokkaan. Ohjelmoijalle jää samalla vastuu siitä mitä funktio tosiasiassa tekee. Rajapintojen funktiot ovat yleensä tyypiltään takaisinkutsuttavia.

Rajapinta liitetään luokkaan varatulla sanalla **implements**. Tällöin luokka sisällyttää rajapinnan funktiot ja muuttujat, mutta kaikki funktiot on ylikirjoitettava - muuten kääntäjä antaa virheilmoituksen. Rajapintoja voidaan sisällyttää luokkaan yksi tai useampia. Rajapintojen nimet kerrotaan implements-sanan jälkeen pilkulla erotettuna.

```
class luokka implements Rajapinta, Rajapinta2
{
 public void funktio();
}
```

Useiden rajapintojen liittäminen luokkaan on moniperiytyvyyttä. Se onkin ainoa tapa toteuttaa moniperiytyvyyttä Javassa.

Seuraavaksi tutustutaan siihen, kuinka rajapintoja tehdään itse. Edellä todettiin, että rajapinta on kuin luokka, mutta se ei sisällä funktioiden ohjelmakoodia, ainoastaan esittelyn. Rajapinnassa on vakioarvoiset muuntimet käytössä: rajapinnat ovat joko private- tai public-tyyppisiä, rajapinnan funktiot ovat aina public- ja abstract-tyyppisiä. Sen vuoksi funktioilla ei voi olla vartaloa. Muuttujat ovat tyypiltään public, static ja final. Tämä määrittely estää muun muassa muuttujien arvon muuttamisen. Muuttujilla pitää olla myös määriteltynä jokin arvo.

Rajapinta määritellään varatulla sanalla **interface** seuraavasti:

```
public interface Rajapinta
{
  int muuttuja = 1;
  void funktio();
}
```

Rajapinta voi yhdistää yhden tai useamman rajapinnan. Toimintoon tarvitaan varattua sanaa extends.

```
public interface Rajapinta extends Rajapinta2
{
  int muuttuja = 1;
  public void funktio();
}
```

Rajapinnat voivat olla tietotyyppeinä kuten luokatkin. Määrittely tapahtuu seuraavasti ilman new-operattorin käyttöä:

```
rajapinnanNimi  muuttujanNimi
```

3.26 Luokkakirjastot

Java-kieli tuntee C++-kielen tapaisen ohjauskomennon, jolla valmiita luokkia sisällytetään ohjelmaan. Komento on nimeltään **import**, ja se vastaa C++-kielen include-komentoa. Valmiit luokat sisältävät tietenkin runsaasti monipuolisia funktioita, joilla tehdään haluttuja toimenpiteitä, kuten tulostetaan tekstiä ja grafiikkaa, soitetaan musiikkia ja niin edelleen. Luokkien sisältämät funktiot ovatkin ainoa keino saada ohjelma tekemään jotakin järkevää, koska ei ole muuta tapaa osoittaa tietokoneen resursseja. Java ei ole aivan yhtä koneläheinen ohjelmointikieli kuin C++.

Luokkakirjastot, jotka halutaan liittää ohjelmaan, kerrotaan heti lähdekoodin alussa. Kaikkia luokkakirjastoja ei ole pakko liittää, ei myöskään kaikkia luokkakirjaston luokkia (luokkakirjasto voi sisältää useita luokkia). Jos halutaan liittää vain yksi luokka, menetellään näin:

```
import java.awt.Graphics;
```

Jos taas halutaan liittää saman luokkakirjaston kaikki luokat (tässä tapauksessa kirjaston java.awt) menetellään näin:

```
import java.awt.*;
```

Javassa luokkakirjastoja kutsutaan paketeiksi (*packages*). Paketit sisältävät luokkia ja rajapintoja. Paketit eivät koskaan sisällä luokkien ulkopuolella olevia funktioita tai muuttujia. Exception luokat ovat tarkoitettu poikkeustilanteiden hallitsemiseen. Luokkakirjastoja on 22 kappaletta ja sen lisäksi kolme Sunin omaa luokkaa, joita ei ohjelmoinnissa tarvita.

Java API Packages

package java.applet
package java.awt
package java.awt.datatransfer
package java.awt.event
package java.awt.image
package java.beans
package java.io
package java.lang
package java.lang.reflect
package java.math
package java.net
package java.rmi
package java.rmi.dgc
package java.rmi.registry
package java.rmi.server
package java.security
package java.security.acl
package java.security.interfaces
package java.sql
package java.text
package java.util
package java.util.zip

Other Packages

package sun.tools.debug
package sunw.io
package sunw.util

Paketti java.applet on ohjelmien perusluokka. Jokainen appletti perustuu Applet-luokkaan. Luvussa 5 perehdytään luokan käyttöön tarkemmin.

Paketti java.awt on GUI:n (*Graphics User Interface*) eli käyttöliittymän perusluokka. Java.awt on ehdottomasti eniten käytetty paketti. Sen sisältämiä luokkia tarvitaan lähes kaikissa kirjan esimerkeissä.

Paketti java.awt.Event on tapahtumien perusluokka. Luokka sisältää kaikki mahdolliset tapahtumat, joita käyttäjä voi aiheuttaa. Pakettiin tutustutaan luvussa 6.

Paketti java.awt.image on tarkoitettu GIF- ja JPG-kuvien käsittelyyn. Sen avulla voidaan tehdä itse kuvia ja tulostaa piirtopinnalle pikseli pikseliltä. Pakettiin tutustutaan luvussa 12.

Paketti java.io on peruspaketti I/O-toimintoihin. Paketissa on valmiit I/O-toiminnot, tosin ei yhtä laajat kuin C:ssä. Valitettavasti WWW-selaimet eivät salli I/O-toimintojen käyttöä verkossa. Paketin käyttöön tutustutaan luvussa 4.

Paketti java.lang sisältää tietotyyppejä vastaavat oliot. Täällä sijaitsee myös luvun alussa mainittu String-luokka. Suora tyyppimuunnos näiden luokkien ja tietotyyppien välillä ei ole mahdollista. Tämän luokkakirjaston luokkiin ja niiden funktioihin perehdytään tämän luvun lopussa.

3.27 Luokkakirjasto java.lang

Edellä on jo mainittu, että luokkakirjasto java.lang sisältää tietotyyppejä vastaavat oliot. Niitä ovat Boolean, Character, Double, Float, Integer ja Long. Seuraavaksi on tarkoitus tutustua Integer-luokkaan ja sen tärkeimpiin funktioihin. Muiden luokkien funktiot ovat toiminnaltaan samanlaisia. Katsotaan, kuinka tapahtuu tyyppimuunnos oliosta toiseen ja oikeisiin (yksinkertaisiin) tietotyyppeihin.

```
Integer i = new Integer(”1”);

Integer i = new Integer(1);
```

Integer-luokassa on seuraavat funktiot, jotka palauttavat olion arvon halutussa tietotyypissä.

intValue()	Olion arvo tietotyypiksi **int**
longValue()	Olion arvo tietotyypiksi **long**
doubleValue()	Olion arvo tietotyypiksi **double**
floatValue()	Olion arvo tietotyypiksi **float**

Merkkijonojen (*String*) muuntaminen Integer-tyypiksi ja päinvastoin on helppoa.

toString()	Integer merkkijonoksi
getInteger(String)	Merkkijono Integeriksi

Jos merkkijono halutaan muuttaa int-tietotyypiksi, pitää ensin tehdä merkkijonosta Integer-tyyppi **`getInteger`**-funktiolla ja sen jälkeen palauttaa **`intValue`**-funktiolla int-tietotyyppi. Tämän kaltaista toimintaa tarvitaan, kun ohjelmalle annettuja parametreja muutetaan merkkijonoista numeroiksi.

```
int luku = Integer.getInteger(merkkijono).intValue();
```

On olemassa vielä yksi tärkeä luokka, jolla suoritetaan matemaattisia toimenpiteitä - se on tietenkin **Math**. Java-kielihän ei sisällä montakaan operaattoria, joilla suoritetaan laskutoimituksia. Monimutkaisemmat laskutoimitukset, kuten neliöjuuri ja potenssiin korotus, saadaan Math-luokan avulla. Esimerkiksi kun halutaan saada neliöjuuri luvusta 4 menetellään seuraavasti:

```
int a = Math.sqrt(4);
```

Tai kun halutaan korottaa luku 4 potenssiin 2:

```
int a = Math.pow(4,2);
```

Näiden funktioiden lisäksi Math-luokassa on monia muitakin hyödyllisiä funktioita. Niistä muutamia on esitelty seuraavassa.

abs()	Absoluuttinen arvo
acos()	Arkuskosini välillä 0.0 ja pii
asin()	Arkussini välillä -pii/2 ja pii/2
atan()	Arkustangentti välillä -pii/2 ja pii/2
cos()	Kosini
sin()	Sini
tan()	Tangentti
exp(a)	e (2,718) potenssiin a
log()	Luonnollinen logaritmi
max(a,b)	Suurempi luvuista
min(a,b)	Pienempi luvuista
pow(a,b)	a potenssiin b
random()	Satunnaisluku
sqrt()	Neliöjuuri

Java.lang-luokkakirjasto sisältää lisäksi yhden tärkeän luokan - **System**. System-luokka on pääsytie tietokoneen käyttöjärjestelmään. Sieltä löytyy muun muassa muuttujat IN ja OUT, jotka ovat pääsytie tulostevirtoihin. Käytännöllinen funktio on myös exit(int), joka lopettaa Java-ohjelman suorittamisen. System-luokan funktiot eivät ole käytettävissä Netscapessa.

3.28 Poikkeustilanteiden käsittely

Poikkeustilantaiden (*exception)* käsittelyrutiinit ovat yksi osa Javan monipuolista virheentarkastustoimintaa. Exception-rutiinit tarkkailevat ajon aikana sattuneita poikkeustilanteita, joita ei voi mitenkään ennalta arvioida käännösvaiheessa. Yksi tyypillinen poikkeustilanne on alustamattoman olion käyttö (NullPointerException). Tietenkään kaikkia virhetilanteita ei voi käsitellä, jolloin selain saattaa kaatua.

Kielessä on kaksi rakennetta, joilla poikkeustilanteita ohjataan: **try catch** ja **try finally**. Poikkeustilanteita varten kirjoitetaan aaltosulkujen sisään poikkeustilanteenkäsittelijä, yleensä viesti siitä, että poikkeustilanne on tapahtunut. Tietenkin rutiinissa voidaan yrittää korjausta, esimerkiksi alustamalla muuttuja johonkin arvoon.

try catch-lauseen try-haarassa kirjoitetaan ne toimenpiteet, jotka voivat aiheuttaa poikkeustilanteen. Catch-haarassa määritellään ne poikkeustilanteet, joihin reagoidaan. Yksi catch nappaa yhden poikkeustilanteen, useammalla catch-haaralla napataan useampia. Mikäli try-haarassa tapahtuu poikkeustilanne, keskeytetään lauseiden suorittaminen ja hypätään catch-haaraan. Catch-haarasta ei päästä enää takaisin try-haaraan.

```
try {
     int luku = String.parseInt("196");
    }
 catch (NullPointerException n)
    { system.out.println("virhe!"); }
 catch (Exception e)
    { system.out.println("virhe!"); }
```

try finally-lauseessa try-haaraan kirjoitetaan ne toimenpiteet, jotka voivat aiheuttaa poikkeustilanteen. Finally-haarassa määritellään ne lauseet, jotka suoritetaan riippumatta siitä, tuliko poikkeustilanne vai ei. Finally-haara suoritetaan myös, vaikka try-haara sisältäisi return-, continue- tai throw-lauseet.

```
try {
  int luku = String.parseInt("196");
}
finally {
  system.out.println("Morjens!");
}
```

Throw-lauseen tarkoituksena on generoida uusi poikkeustilanne, lähinnä käyttäjän oma poikkeustilanne. Ensiksi luodaan oma luokka, joka periytyy Exception-luokasta.

```
class Poikkeustilanne extends Exception
```

```
{
}
```

Sen jälkeen voidaan throw-lauseen avulla luoda uusi poikkeustilanne. Poikkeustilanteen nappaamiseksi käytetään edellä esiteltyä try-catch-lausetta.

```
try
{
  if (a !=1)
    print (”ok”);
  else
   throw new Poikkeustilanne();
}
catch (Poikkeustilanne v)
{
}
```

Javassa on valmiina useita luokkia, joista kukin vastaa omasta poikkeustilanteestaan, tosin yksikään niistä ei tee muuta kuin tulostaa tapahtuman nimen. Alla olevassa listassa on esitelty muutamia ajon aikaisia poikkeustilanteita. API:a katsomalla löytää ne funktiot, jotka tarvitsevat poikkeuksenkäsittelijää. Try-catch- ja try-finally-lauseita tarvitaan vain silloin, kun funktio tarvitsee poikkeuksen hallintaa.

ArithmeticException

- Nollalla jako

ArrayIndexOutOfBoundsException

- Alkiota ei ole olemassa

ClassNotFoundException

- Luokkaa ei ole olemassa

NegativeArraySizeException

- Negatiivinen taulukon koko

NullPointerException

- Oliota ei ole olemassa

4. Sovellusohjelmointi

Tässä luvussa tutustutaan Java-sovelluksiin (*application*), jotka voidaan ajaa joko Java-tulkin päällä tai suoraan käyttöjärjestelmässä, mikäli Java-tuki on olemassa. Sovellukset eivät tarvitse lainkaan WWW-selainta.

Sovelluksen ja appletin ero koodillisesti on minimaalinen (kuten luvussa 5 huomataan), mutta ajettaessa ero onkin huomattava. *Sovelluksella on enemmän oikeuksia kuin appletilla.* Esimerkiksi sovellusohjelma voi kirjoittaa ja lukea käyttäjän kiintolevyä, suorittaa verkko-operaatioita eri protokollilla mihin koneisiin tahansa.

4.1 Tietovirrat

Tähän mennessä sovellukset ovat tulostaneet System-luokan avulla jotakin yksinkertaisia merkkijonoja ruudulle. Lähes tietämättämme olemme käyttäneet avuksi yhtä standardia tietovirtaa - out. Käytännössä out on luokka, joka sisältää funktoita (esim. println) ja muuttujia.

System-luokassa määritellään muitakin tietovirtoja kuin out. Ne ovat in ja err. Nämä tietovirrat ovat tarkoitettu syöttötietojen lukemiseen ja virheiden tulostamiseen.

out	Standardi tulostusvirta. Oletusarvo näyttö.
in	Standardi syöttövirta. Oletusarvo näppäimistö
err	Standardi virhetulostusvirta. Oletusarvo näyttö

Tietovirtojen uudelleenohjaus on myös mahdollista. Näin voidaan estää esimerkiksi virheilmoitusten tulostuminen sovelluksen käynnistäneeseen ikkunaan tai lukea tietoa näppäimistön sijasta jostakin tiedostosta. Tietovirtojen uudelleenohjaus tapahtuu seuraavilla System-luokan funktioilla.

setErr(PrintStream)	Uudelleenohjaa virhetulostusvirran
setIn(InputStream)	Uudelleenohjaa syöttövirran
setOut(PrintStream)	Uudelleenohjaa tulostusvirran

Kaikki nämä standardit tietovirrat ovat määritelty jonkin java.io-luokasta löytyvän ”virtaluokan” olioiksi. Esimerkiksi out on PrintStream-luokan olio. PrintStream tarjoaa esimerkiksi tutun

println-funktion, jolla ruudulle on tulostettu tekstiä. PrintStream-luokan tärkeimpiä funktioita ovat:

print()	Tulostaa ilman rivinvaihtomerkkiä
println()	Tulostaa lopuksi rivinvaihtomerkin
write()	Tulostaa tavuja halutun määrän halutusta kohdasta

Standardia syöttövirtaa ei ole tähän mennessä käytetty lainkaan. Syöttövirran avulla voidaan näppäimistöltä antaa ohjelmaan tekstiä, lukuja jne. In-olio ei kuitenkaan osaa lukea näppäimistöltä merkkejä vaan tavuja. Tarvittaessa tavut on muutettava String-luokan avulla merkkijonoksi.

mark(int)	Merkitsee nykyisen kohdan
markSupported()	Testaa onko merkitseminen mahdollista
read()	Lukee seuraavan tavun
read(byte[])	Lukee useita tavuja
reset()	Asettaa sijainnin viimeksi merkittyyn kohtaan
skip(long)	Hyppää yli annetun määrän tavuja

Esimerkki print- ja read-funktioiden käytöstä.

```
import java.io.*;

class App1
{

  public static void main(String args[])
  {
    byte bytes[] = new byte[100];
    try {
     System.out.print("Kirjoita nimesi: ");

     System.in.read(bytes);

     String s = new String(bytes);

     System.out.println("Nimesi on "+s);
     } catch (IOException io)
```

```
        {
        }
    }
}

Tuloste:    Anna nimesi: Petri
            Nimesi on Petri
```

PrintStream- ja InputStream-luokat, joita käyttävät oliot out ja in, eivät sisällä juuri lainkaan funktioita virtojen käsittelyyn. Javassa on päädytty sellaiseen rakenteeseen, että tarvittavien ominaisuuksien (esim. lukupuskuri, jolla estetään tiedon katoaminen) lisääminen onnistuu luokkien avulla. Yhdessä luokassa ei ole siis kaikkia ominaisuuksia, koska kaikkia ominaisuuksia ei välttämättä tarvita. Esimerkiksi puskuroitu tulostevirta saadaan aikaiseksi luokan BufferedInputStream avulla. Puskuri muodostetaan muodostimella siten, että annetaan parametrina InputStream-olio ja puskurin koko (ei välttämätön).

```
BufferedInputStream buffer = BufferedInputStream(in);
```

4.2 Muita System-luokan funktioita

System-luokka sisältää paljon käyttökelpoisia funktioita, joita voi käyttää vain sovelluksissa. Tutustutaan seuraavaksi muutamiin.

System-luokan **`exit`**-funktiota käytetään lopettamaan ajossa oleva Java-sovellus. Funktiota kutsutaan parametrinaan kokonaisluku. Tyyppillisesti luku 0 tarkoittaa normaalia lopetusta ja nollasta poikkeava epänormaalia lopetusta. Exit-funktiolla voidaan esimerkiksi lopettaa ohjelman suorittaminen virheen sattuessa.

```
if (a != 0)
 System.exit(1);    // VIRHE!! Lopeta ohjelma
```

Joskus tarvitaan tietoa siitä missä hakemistosta Java-sovellus on käynnistetty, tai minkä käyttöjärjestelmän alla sovellusta ajetaan. Nämä tiedot saadaan luettua funktiolla **`getProperty`**. Funktio palauttaa tiedon merkkijonossa. Seuraavaan taulukkoon on koottu ominaisuudet, jotka getProperty ymmärtää.

Esimerkiksi, kun luetaan käyttäjän nykyinen työhakemisto, josta sovellusohjelma käynnistettiin, menetellään seuraavasti:

```
String hakemisto = System.getProperty("user.dir");
```

Jos halutaan lukea samasta hakemistosta tiedosto (esim. ohjelman alustustiedosto), josta ohjelma käynnistettiin, voidaan hyödyntää ominaisuutta file.separator. **`GetProperty`**-funktio palauttaa tällöin käyttöjärjestelmäkohtaisen tiedostopolun erottimen.

```
String ini = System.getProperty(”user.dir”)+
             System.getProperty(”file.separator”)+
             ”menu.ini”;;
```

Ominaisuus	Selitys
java.version	**Javan versionumero**
java.vendor	**Javan tuottaja**
java.vendor.url	**Javan tekijän URL-osoite**
java.home	**Javan asennushakemisto**
java.class.version	**Javan luokkien versionumero**
java.class.path	**Javan luokkien polku**
os.name	**Käyttöjärjestelmän nimi**
os.arch	**Käyttöjärjestelmän arkkitehtuuri**
os.version	**Käyttöjärjestelmän versionumero**
file.separator	**Tiedostoerotin**
path.separator	**Käyttöjärjestelmän polun erotin**
line.separator	**Rivierotin ("\n")**
user.name	**Käyttäjänimi**
user.home	**Käyttäjän kotihakemisto**
user.dir	**Käyttäjän nykyinen työhakemisto**

4.3 I/O-toiminnot

Tiedostojen käsittelyyn käytetään **java.io**-luokkakirjaston luokkaa **File**. File-luokka tarjoaa kattavat ominaisuudet tiedostojen käsittelyyn. Näitä ominaisuuksia ovat muun muassa seuraavat:

- Tiedostoon kirjoitus
- Tiedoston luku
- Tiedoston tuhoaminen

- Tiedostoston luonti
- Tiedoston nimeäminen
- Tiedoston sisällön vertaaminen
- Tiedoston polun luku
- Tiedoston testaus (onko tiedosto olemassa, onko se alihakemisto, milloin viimeksi muutettu)
- Tiedoston pituuden luku
- Hakemiston tulostaminen
- Hakemiston tuhoaminen
- Hakemiston luonti

HUOM! Kaikki nämä ominaisuudet ovat käytettävissä sovelluksissa, mutta yksikään niistä ei toimi selaimessa ajettavassa appletissa. Estäminen on tehty turvallisuussyistä! Java1.1 tarjoaa kuitenkin mahdollisuuden I/O-ominaisuuksien suorittamiseen selaimessa, jos appletti on sähköisesti allekirjoitettu ja käyttäjä antaa luvan kyseisiin operaatioihin.

Tiedostojen käsittelyssä Java on käyttöjärjestelmästä riippumaton. Erottimena käytetään Unix-käyttöjärjestelmästä tuttua kauttaviivaa. Periaatteena on ollut tuottaa käytäntö, joka sopii kaikille ohjelmointiympäristöille.

File-luokalla on muodostimia kolme ja ne ovat:

File (String)	Tiedoston nimi (suhteellinen menetelmä)
File (String, String)	Tiedoston polku ja nimi (absoluuttinen menetelmä)
File (File, String)	Tiedoston polku ja nimi (absoluuttinen menetelmä)

Tiedosto nimi voi olla suhteellinen tai absoluuttinen: suhteellinen tiedoston nimi tarkoittaa sitä, että se on annettu ilman polkua. Absoluuttinen puolestaan tarkoittaa sitä, että tiedoston nimi on annettu tarkalleen mahdollisine hakemistoineen. File-olio edustaa tiedoston nimeä siinä tietokoneessa, jossa sovellusohjelma ajetaan.

```
File tiedosto = new File ("tiedostonnimi");
```

Muodostamisen jälkeen File edustaa pelkästään oliota, ei todellista kiintolevyllä olevaa tiedostoa - olion muodostamisen onnistuminen ei takaa, että kyseinen tiedosto on olemassa. Tieto siitä, onko tiedosto olemassa, saadaan funktiolla **`exists`**.

```
if (tiedosto.exists())
  // Tiedosto on olemassa
else
  // Tiedostoa ei ole olemassa
```

Tiedostoon voidaan kohdistaa myös muitakin testejä. Esimerkiksi tieto siitä, voidaanko tiedostoa lukea saadaan **`canRead`**-funktiolla tai voidaanko siihen kirjoittaa saadaan **`canWrite`**-funktiolla.

```
if (tiedosto.canRead())
 // luku mahdollista
if  (tiedosto.canWrite())
 // kirjoittaminen mahdollista
```

Tiedoston lukuun ja kirjoittamiseen tarvitaan puskuri. Data luetaan puskuriin (InputStream) tai se kirjoitetaan puskurista (OutputStream). Hyödynnetään tiedostopuskureita **FileInputStream** ja **FileOutputStream**, jotka sisältävät tiedoston luku- ja kirjoitusfunktiot.

```
FileInputStream  fi = new FileInputStream(File);
FileOutputStream fo = new FileInputStream(File);
```

Puskureissa FileInputStream ja FileOutputStream on valmiina funktiot, joilla luetaan ja kirjoitetaan tietoa tiedostoon. Seuraavassa kaksi ensimmäistä funktiota kuuluvat FileInputStream-luokalle ja kaksi jälkimmäistä FileOutputStream-luokalle.

read()	Lukee tavun
read(byte[])	Lukee useita tavuja
write()	Kirjoittaa tavun
write(byte[])	Kirjoittaa useita tavuja

Hakemiston käsittely File-luokan avulla on myös mahdollista. Hakemistossa olevat tiedostot voidaan listata **`list`**-funktiolla ja hakemistoja voidaan luoda **`mkdir`**-funktiolla. Hakemistoja voidaan myös nimetä **`renameTo`**-funktiolla. Seuraavat funktiot löytyvät File-luokasta, ja niitä voidaan käyttää heti, kun File-olio avataan.

list()	Tulostaa hakemiston tiedostot
list(Filter)	Tulostaa hakemistosta tietyn päätteiset tiedostot
mkdir()	Luo hakemiston
renameTo(File)	Nimeää hakemiston/tiedoston

I/O-kirjastossa on lisäksi toisenlainen File-luokka nimeltään **RandomAccessFile**. Se sisältää valmiiksi useita näppäriä funktioita, joilla tiedostoa voidaan lukea tai siihen voidaan kirjoittaa. Erillistä luku/kirjoituspuskuriakaan ei tarvita.

```
RandomAccessFile rf = new
                      RandomAccessFile(file,mode);
```

Tiedosto luodaan muodostimella, jonka parametrina on File-olio (tiedoston nimi) ja String-olio (moodi). Moodi on "r" (luku) tai "rw" (luku/kirjoitus).

```
byte puskuri = new byte[256];

rf = new RandomAccessFile(file,"r");
rf.read(puskuri);
```

4.4 Ohjelmien suorittaminen

Java-sovellus voi käynnistää minkä tahansa tietokoneessa olevan ohjelman, esimerkiksi javac-kääntäjän, Netscape Navigator -selainohjelman tai vaikkapa Word-tekstinkäsittelyohjelman. Tämä tapahtuu kutsumalla luokkakirjaston java.lang luokan **Runtime** funktiota **`exec`**. Ylikuormitettuja exec-funktiota on käytettävissä kaksi kappaletta. Ne eroavat toisistaan ohjelmaan syötettävien parametrien syöttötavasta.

exec(String)	Suorittaa ohjelman. Parametrit syötetään yhdessä merkkijonossa välilyönnillä erotettuna.
exec(String[])	Suorittaa ohjelman. Parametrit syötetään erillisissä merkkijonoissa.

Exec-funktio käynnistää ohjelman omassa erillisessä prosessissa ja palauttaa suorituksen funktiota kutsuneeseen ohjelmaan heti prosessin käynnistymisen jälkeen. Exec-funktio palauttaa **Process**-olion, jonka avulla prosessin tilaa voidaan ohjata.

Exec-funktiota ei voi kutsua suoraan luokan nimen avulla (Runtime.exec). Runtime-olion muodostaminen ei myöskään onnistu. Nämä johtuvat siitä, että exec-funktio staattinen ja Runtime-luokalta puuttuu muodostin. Siksi tarvitaan avuksi funktio **`getRuntime`**, joka palauttaa Runtime-luokan esiintymän.

```
Process pr = Runtime.getRuntime().
             exec("javac"+filename);
```

Käynnistetyn ohjelman tulosteet eivät tule näkymään missään ikkunassa ellei tietovirtoja uudelleenohjata. Uudelleenohjaus tapahtuu tässä tapauksessa Process-luokan, eikä System-luokan funktioilla. Varsinaisen uudelleenohjauksen suorittaa ohjelmoija siten, että hän ohjaa tietovirran haluttuun olioon. Apuna käytetään funktioita, jotka palauttavat nykyisen tietovirran.

getErrorStream()	Palauttaa virhetulostusvirran (InputStream)
getInputStream()	Palauttaa syöttövirran (InputStream)
getOutputStream()	Palauttaa tulostevirran (OutputStream)

Esimerkiksi virhetulostusvirran ohjaaminen omaan BufferedInputStream-olioon tapahtuu seuraavasti:

```
BufferedInputStream input =
          newBufferedInputStream(pr.getErrorStream());
```

Jos tuloste halutaan näyttää välittömästi prosessin suorittamisen jälkeen, voidaan odottaa prosessin suorituksenn loppumista. Se toteutetaan funktiokutsulla **`waitFor`**.

```
pr.waitFor();
```

4.5 Ikkunointia

Tähän mennessä on sovelluksissa käytetty vain merkkipohjaista komentorivitilaa. Joissakin tapauksissa ohjelmointi merkkipohjaisessa tilassa on hyödyllistä, mutta nykyään se alkaa olla vanhanaikaista. Ohjelmointi graafisessa tilassa ei ole vaikeampaa kuin merkkipohjaisessa. Sen sijaan käytettävissä olevat resurssit lisääntyvät huomattavasti. Voidaan hyödyntää ikkunoita, valikoita sekä graafisia elementtejä ja grafiikkaa. Lisäksi valmiiden avaa- ja tallenna-dialogien sekä tulosta-dialogien avulla tiedostojen käsittely ja tulostaminen helpottuvat huomattavasti.

Merkkipohjaisesta tilasta siirtyminen graafiseen tilaan tapahtuu yhdellä ohjelmarivillä - avataan ikkuna (*frame*). Ikkunaan ei kuitenkaan voi tulostaa perinteisillä system-kirjasto out-olion

funktioilla. (Ikkunan avaamisen jälkeen out-luokan println-funktio tulostaa edelleen komentoriville. Tätä ominaisuutta voi hyödyntää esimerkiksi ohjelman debuggaukseen.) Tutustutaan seuraavaksi mahdollisimman lyhyeen graafisessa käyttöjärjestelmässä toimivaan esimerkkiin. Esimerkissä avataan ikkuna, annetaan sille nimeksi ”ikkunointia” ja asetetaan ikkunan kooksi 300*300.

```
import java.awt.*;

public class gfxsovellus
{
 public static void main(String args[])
 {
  Frame f = new Frame("Ikkunointia");
  f.setVisible(true);
  f.setSize(300,200);
 }
}
```

Yllä olevalla ohjelmarungolla on joitakin huonoja puolia. Koska main-funktio on staattinen, se ei voi esimerkiksi käsitellä luokan muita ei-staattisia muuttujia (tosin tässä tapauksessa niitä ei ole). Sellaisessa rakenteessa, jonka main-funktiossa luodaan ohjelman olio, mahdollistaa luokan muiden muuttujien käytön. Kun luokka lisäksi periytetään Frame-luokasta, saadaan ikkunoitu sovellusohjelma helposti käyttöön. Tässä esimerkissä ikkunan ominaisuuksia ei tarvitse käyttää f-olion kautta, koska luokka itse edustaa ikkunaa.

```
import java.awt.*;

public class gfxsovellus1 extends Frame
{
 public static void main(String args[])
 {
 gfxsovellus1 sovellus = new gfxsovellus1();
 }

 public gfxsovellus1()
 {
  super("Ikkunointia");
  setVisible(true);
  setSize(300,200);
 }
}
```

Kun kokeilet näitä sovelluksia ja yrität tuhota ikkunan, huomaat jotakin outoa: ikkuna ei tuhoudukaan. Tämä johtuu siitä, että tällaista tapahtumaa ei ole ohjelmoitu sovellukseen. *Luvussa*

8 Ikkunat, valikot ja dialogit tulet tutustumaan kuinka tällainen tapahtumakäsittelijä ohjelmoidaan.

HUOM! Vaikka luvusta 6 *Interaktiivisuus* lähtien kirjan esimerkit ovat kirjoitettu appleteiksi, voidaan samoja ominaisuuksia käyttää myös sovelluksissa. Kaikki appletit voidaan kirjoittaa sovelluksiksi pienillä muutoksilla. (Katso luku 4.8.)

4.6 Liityntä käyttöjärjestelmään

Appletteja tehtäessä ohjelmoijan ei tarvitse välittää miten Java liittää itsensä käyttöjärjestelmään. Selaimen päällä kaikki tapahtuu automaattisesti ja liityntä on jo valmiina, kun appletti avautuu. Sovellusohjelmoinnissa pitää huomioida se, että Java-sovellus ei ole välttämättä saanut käyttöönsä liittymää käyttöjärjestelmään. Tätä liittymää kutsutaan Javassa nimellä **peer**.

Olet saattanut huomata, että awt-luokkakirjaston luokilla on takaisinkutsuttavafunktio **`addNotify`**. Se kertoo, milloin liityntä on luotu. Käytännössä siis ylikirjoittamalla tämä funktio ja kirjoittamalla koodi funktion sisään, voidaan varmistaa, että ohjelmakoodi toimii niinkuin sen pitääkin.

4.7 Useasta luokasta koostuva sovellus

Olio-ohjelmoinnissa ohjelmat koostuvat useista luokista. Luokat ovat "ohjelmakomponentteja", jotka kätkevät taakseen jonkun toiminnallisuuden. Esimerkiksi sovellus voi koostua main-funktion sisältävästä pääohjelmasta, ikkunaluokasta sekä muista erillisistä toiminnoista, esimerkiksi dialogeista, viestilaatikoista. Kun ohjelma koostuu useasta luokasta, syntyy monesti myös siitä ongelmia. Mihin tuloste syötetään, kuka kutsui luokkaa jne. Varsinkin, kun Java 1.1 -versiossa tapahtumat voidaan delegoida mihin luokkaan tahansa, voi tällaisia ongelmia syntyä helposti.
Luokan sisällä voidaan käyttää muuttujaa (muuttuja voi olla myös olio), joka näkyy luokan kaikille funktioille. Mutta tällainen muuttuja ei kuitenkaan näy muille luokille. Javassa ei ole myöskään mahdollisuutta käyttää globaaleja-muuttujia, jotka näkyisivät kaikissa luokissa. Onglma on siis se, miten saadaan jokin muuttuja näkymään halutulle luokalle. Ratkaisuja on ainakin kaksi.

- Syötetään muodostimessa parametrina tarvittava olio ja tallennetaan olio luokan muuttujaan.
- Syötetään funktion parametrina tarvittava olio

4.8 Sovellukset appletteina

Tietyin ehdoin sovellukset voivat toimia myös appletteina (jos appletin rakenne on outo, voit lukea siitä luvusta 5). Helpointa muutos on silloin, kun kirjoitusvaiheessa on pidetty appletiksi muuttaminen mahdollisena. Tärkein seikka muutoksessa on se, että appletti on hyvin rajoittunut sovellukseen nähden. Turvallisuussyistä useat toiminnot ovat estetty.

Appletit eivät voi

- Käyttää tekstitilaa
- Käyttää System-luokkaa
- Käyttää File-luokkaa
- Käyttää Runtime-luokkaa
- Käyttää main-funktiota

Myöhemmin tulet oppimaan myös muita asioita, joita appletit eivät voi käyttää. Ominaisuuksien puuttuminen vaikeuttaa sovelluksen kääntämistä appletiksi. Toisinpäin on helpompaa, koska sovellus voi toteuttaa kaikki samat asiat kuin appletti.
Muuttaminen appletiksi tapahtuu helpoimmin, jos sovellus käyttää tietynlaista rakennetta. Esimerkiksi edellinen esimerkki **gfxsovellus1.java** on aika vaikea muuttaa appletiksi, koska sovellus koostuu yhdestä luokasta. Samanaikaisesti ei voi periyttää sekä Frame- että Applet-luokista. Jos sovellus koostuu alkujaan kahdesta luokasta, joista toinen on erillinen ikkunaluokka, on muutos helpoin toteuttaa.

```
import java.awt.*;

public class GfxSovellus2
{
 public static void main(String args[])
 {
  Gfxsov2Ikkuna sovellus = new gfxsov2Ikkuna();
 }
}

class Gfxsov2Ikkuna extends Frame
{
 public Gfxsov2Ikkuna()
 {
  super("Ikkunointia");
  setVisible(true);
  setSize(300,200);
 }
```

```
}
```

Sovellus **GfxSovellus2.java** muutettuna Java-appletiksi. Nyt gfxsovellus2 periytetään Applet-luokasta. Init-funktiossa kutsutaan ikkunaluokkaa samalla tavalla kuin main-funktiossa. Ohjelman toiminnallisuus on sijoitettu luokkaan **Gfxsov2Ikkuna**.

```
import java.applet.*;
import java.awt.*;

public class GfxSovellus2 extends Applet
{
 public void init()
 {
  Gfxsov2Ikkuna sovellus = new Gfxsov2Ikkuna();
 }



 public static void main(String args[])
 {
  Gfxsov2Ikkuna sovellus = new Gfxsov2Ikkuna();
 }
}

class Gfxsov2Ikkuna extends Frame
{
 public Gfxsov2Ikkuna()
 {
  super("Ikkunointia");
  setVisible(true);
  setSize(300,200);
 }

}
```

5. Appletin rakenne

Edellisissä luvuissa on jo kerrottu Java-sovellusten taustaa. Tässä luvussa perehdytään siihen, kuinka Java-ohjelma rakennetaan WWW-selaimelle ymmärrettävään muotoon.

5.1 Appletit

Appleteiksi kutsutaan sellaisia Java-ohjelmia, jotka toimivat WWW-selaimessa, esimerkiksi Netscapessa. Applet-nimitys tulee siitä, että Java-ohjelma perustuu Applet-nimiseen luokkaan. Kun vielä HTML-kielessä on lisäksi <APPLET>-tagi, onkin oltava tarkkana, ettei näissä käsitteissä mene sekaisin. On siis olemassa kolmenlaisia asioita, joita sana applet voi tarkoittaa.

- applet = WWW:ssä toimiva Java-ohjelma
- applet = Java-ohjelmien perusluokka
- applet = HTML-kielen tagi

Appleteista kannattaa kirjoittaa yleismuotoisia, joustavia ja helposti parametreilla ohjattavissa olevia. Applettien ei esimerkiksi kannata olla turhan tarkkoja sijainnistaan. Hyvä olisi, että ne toimisivat mistä URL-osoitteesta tahansa. Joustavia appletit ovat silloin, kun niillä on hyvä parametreilla ohjattavuus - käyttäjän ei tarvitse muokata ja kääntää ohjelmaa uudelleen, jos joitakin ominaisuuksia muutetaan. Esimerkiksi animaatiossa kannattaa sallia kuvien määrän ja esitysnopeuden muuttaminen parametreilla mieluummin kuin uudelleen kääntämisellä.

Seuraavassa on esitetty lyhin mahdollinen appletti, joka tulostaa tekstin Hello WWW. Appletin nimi on **EkaApplet**. Vertaa appletin rakennetta ensimmäiseen Java-sovellukseen EkaOhjelma (esitetty luvussa 2). Kummatkin tulostavat yksinkertaisen teksin.

```
import java.awt.*;
import java.applet.*;

public class EkaApplet extends Applet
{
 public void paint (Graphics g)
 {
  g.drawString(”Hello WWW”, 100, 100);
 }
}
```

5.2 Appletin käyttöönotto

Käännetty applet sisällytetään johonkin HTML-sivuun ennen kuin sitä voidaan käyttää. Testivaiheessa riittää, kun kirjoitetaan yksinkertainen sivu, jossa on vain kielen aloitustagi <HTML>, otsikko <TITLE> sekä <BODY>. Tämän jälkeen voidaan appletti sisällyttää sivuun käyttäen tagia <**APPLET**>. Tagissa pitää määritellä ohjelmakoodi sekä appletille HTML-sivulta varattava koko **width**- ja **height**-arvoilla. Testisivujen kirjoittaminen alkaa olla työlästä, kun testattavia appletteja on useita (vaikka sivut ovatkin yksinkertaisia). Kirjan mukana tuleva editori-ohjelma helpottaa HTML-sivujen tekoa generoimalla valmiiksi testaamiseen sopivan sivun.

HUOM! Appletin koodi on **.class**-päätteinen, ei .java-päätteinen.

Seuraavassa esimerkissä sisällytetään HTML-sivuun appletti nimeltä EkaApplet. Appletin koodi sijaitsee samassa hakemistossa kuin HTML-tiedosto. Appletti varaa itselleen HTML-sivulta 200 pikseliä leveän ja 200 pikseliä korkean alueen.

```
<HTML>
<BODY>
<TITLE> Applet esimerkki </TITLE>
<APPLET code="EkaApplet.class"
  width="200"
  height="200">
</APPLET>
</BODY>
</HTML>
```

Appletille voidaan välittää parametreja <**PARAM**>-tagilla. Se käyttää parametrin nimen ja arvon välittämiseen HTML-kielen standardia kielioppia. Param-tagin **name** kertoo parametrin nimen ja **value** antaa arvon.

HUOM! Appleteille välittyvä tieto on aina merkkijono vaikka value sisältäisikin numeroarvoja.

```
<PARAM  name="parametri"  value="arvo">
```

Esimerkiksi seuraavassa HTML-tiedostossa sisällytetään sivuun Soitto-niminen appletti, jonka tilanvaraus sivulta on 100 * 100 pikseliä. Parametrina *soita* annetaan kappaleen nimi, joka soitetaan.

```
<HTML>
<BODY>
<TITLE> Play </TITLE>
<H1><CENTER> AU-tiedostojen soitto-ohjelma </CENTER></H1><P>
<APPLET code="Soitto.class"  width="100"  height="100">
<PARAM  name="soita"  value=”testi.au">
</APPLET>
</BODY>
</HTML>
```

<APPLET>-tagi voi sisältää muutakin informaatiota kuin edellä on kerrottu. Edellä kerrotut olivat välttämättömiä toiminnan kannalta, mutta seuraavassa taulukossa on esitetty kaikki parametrit.

CODEBASE	URL-osoite, jossa applet sijaitsee.
CODE	Applet, joka suoritetaan
ALT	Teksti, joka tulostetaan, jos applettia ei kyetä ajamaan
NAME	Appletin nimi (vapaaehtoinen)
WIDTH	Leveys, joka varataan appletille
HEIGHT	Korkeus, joka varataan appletille
ALIGN	Tasaus: left, right, top, middle, absmiddle, baseline, bottom ja absbottom
VSPACE	Tyhjä alue, joka jätetään appletin ylä- ja alapuolelle
HSPACE	Tyhjä alue, joka jätetään appletin sivuille

5.3 JAR-tiedostot

JAR-tiedostomuoto (**J**ava **Ar**chive) on käyttöjärjestelmästä riippumaton tiedosto, johon on koottu ja pakattu useita tiedostoja. Käytänössä siis appletti ja sen tarvitsemat tiedostot, kuten kuvat, äänet jne. JAR-tiedostomuoto perustuu tunnettuun zip-tiedostonpakkausmenetelmään.

Tämän uuden tiedostomuodon on etu on se, että yhdellä kertaa voidaan ladata kaikki appletin tarvitsemat tiedostot. Ilman JAR-tiedostoa tarvitaan useita pyyntökutsuja WWW-palvelimelle. Koska JAR-tiedostot ovat lisäksi pakattuja, lataamiseen kuluva aika vähenee huomattavasti. Kun JIT-kääntäjät hidastavat appletin suorituksen aloittamista (tosin lopulta nopeuttavat suorittamista), JAR-tiedostot tuovat tätä turhaan kulunutta aikaa takaisin.

Kolmas etu JAR-tiedostomuodosta on se, että paketin alkuperä voidaan tunnistaa digitaalisen allekirjoituksen avulla (menetelmä johon Microsoftin turvallisuus ActiveX-tekniikassa perustuu). Tunnistamattomat appletit eivät pääse käyttäjän koneeseen ilman lupaa.

Kun JAR-tiedostoa käytetään, pitää applet-tagiin lisätä yksi parametri. Appletin koodin nimeksi (code-parametri) jätetään edelleen appletin nimi, mutta applet-tagiin lisätään uusi parametri **archive**. Parametri ilmaisee, mistä JAR-paketit (erotetaan pilkulla) voidaan ladata käyttäjän koneelle. Esimerkki:

```
<applet code=animator.class
 archive="jars/animator.jar"
 width=460 height=160>
<param name=foo value="bar">
</applet>
```

JAR-paketit tehdään JDK 1.1 -version työkalulla nimeltä **jar**. Syntaksi on seuraava:

```
jar [ optiot ] [manifest-tiedosto] kohde lähde [lähteet]
```

optiot:

c	Luo uuden tiedoston
t	Listaa JAR-tiedoston sisällön
x tiedosto	Purkaa kaikki tai mainitun tiedoston
f	Tekee JAR-tiedoston (jos parametri puuttuu, tiedoston sisältö tulostuu ruudulle)
m	Luo manifest-tiedoston
v	Tulostaa tiedostot, jotka pakettiin otetaan

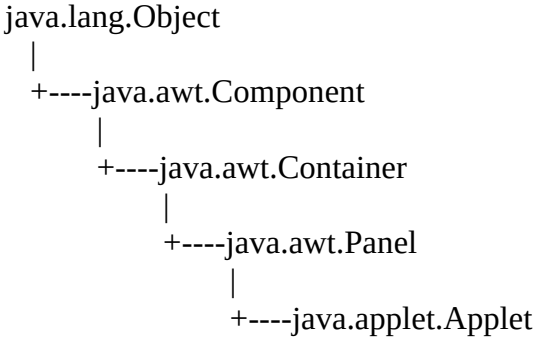
manifest Tiedosto, josta käy ilmi mitkä tiedostot pakettiin alkuperäisesti on kuulunut.

Tyypillisesti jar-käskyä käytetään seuraavasti:

```
jar cf myjarfile *.class
```

5.4 Applet-luokka - ohjelmien perusluokka

Applet on luokka, joka periytyy luokasta Panel. Panel-luokka puolestaan periytyy Container-luokasta, joka periytyy Component-luokasta. Nämä kolme luokkaa ovat GUI:n perusluokkia. Se, mitä niihin sisältyy, selviää vasta luvussa *7. GUI:n perusteet*.



Applet on Java-ohjelmien perusluokka, samoin kuin Object on luokkien perusluokka. Applet-luokka sisältää valmiiksi suuren määrän käyttökelpoisia funktioita ja antaa mahdollisuuden käyttää muiden luokkien funktioita. Omia appletteja tehtäessä on tärkeää sisällyttää Applet-luokka extends-sanalla. Luokan nimi määrää appletin nimen, eli alla appletin nimeksi tulee **Ohjelma**.

```
class Ohjelma extends Applet
{

}
```

Tutustutaan seuraavaksi neljään tärkeään takaisinkutsuttavaan funktioon, jotka appletin pitää sisältää. Nämä takaisinkutsuttavat funktiot antavat ohjelmoijalle perusliittymän applettien hallintaan. Niiden käyttöön ottamiseksi pitää suorittaa kyseisen funktion ylikirjoittaminen omassa luokassa.

Appletin käynnistyessä (ensimmäiseksi) kutsutaan funktiota, jonka nimi on **`init`**. Tämän funktion aikana voidaan muuttaa appletille varattu koko, lukea tiedostoja muistiin animaatiota

varten, vaihtaa appletin taustaväri ja asettaa tarvittavia komponentteja appletin ikkunaan. Esimerkiksi funktiolla **`resize`** voidaan muuttaa appletille HTML-sivulta varattua kokoa.

Init-funktiota kutsutaan vain kerran - silloin kun appletin suoritus aloitetaan ensimmäisen kerran.

```
public void init()
{
 resize(300,300);       // muuta appletin kokoa
}
```

Init-funktion jälkeen kutsutaan **start**-funktiota. Start-funktio nimensäkin mukaisesti aloittaa appletin varsinaisen toiminnan. Start-funktiota voidaan kutsua appletin olemassa olon ajan aikana useita kertoja. Funktiota kutsutaan aina silloin kuin appletin suorittaminen on pysäytetty ja selain tuo appletin uudelleen näkyviin.

Tämän funktion sisään voitaisiin kirjoittaa ohjelman toiminta kokonaan. Toisaalta näin ei voida menetellä, koska koko aikainen start-funktiossa pyöriminen estää käyttöjärjestelmän resurssien käytön, muun muassa tärkeiden viestien jakamisen appletille. Start-funktion varsinainen tarkoitus on aloittaa säikeen (eräänlaisen prosessin) suorittaminen. Säikeessä tehdään yleensä aikaa kuluttavat toimenpiteet. Säikeistä puhutaan lisää luvussa 10.

```
public void start()
{
  Thread t = new Thread(this).start();
}
```

Start-funktion huono puoli on se, että paint-funktio, jota käytetään näytön uudelleen päivittämiseen, kutsutaan vasta start-funktion jälkeen. Tästä aiheutuu sellainen piirre, että komponenttien päivittäminen haluttuun paikkaan move-funktiolla ei onnistu start-funktion aikana. Vaikka näytön päivittämisen vuoksi alla oleva koodi suoritettaisiin start-funktiossa 10-kertaa, ei saavutettaisi haluttua toimintaa.

```
public void start()
{
 for (int i = 1; i < 10; i++)
 repaint();
}
```

Appletin suoritus loppuu, kun applet ei ole näkyvissä. Tällöin kutsutaan automaattisesti **stop**-funktiota. Funktion aikana tulee lopettaa säikeiden suorittaminen ja äänien soittaminen. Stop-funktio ei vielä poista applettia muistista, vaan sen tarkoituksena on informoida ohjelmaa siitä, että appletti ei ole enää näkyvissä. Jos applet tuodaan uudelleen näkyviin, kutsutaan start-funktiota suorituksen aloittamiseksi.

```
public void stop()
{
 thread = null;
}
```

Funktiota **destroy**() kutsutaan, kun applet poistetaan muistista. Tämä on viimeinen mahdollisuus kutsua stop-funktiota ja siten lopettaa prosessit ja musiikin soittaminen.

```
public void destroy()
{
}
```

Näiden takaisinkutsuttavienfunktioiden pohjalta voidaan kirjoittaa yleismuotoinen Java-appletti, joka on perustana tämän kirjan usealle Java-kieliselle esimerkille. Kaikkia näitä takaisinkutsuttaviafunktiota ei tietenkään tarvitse ylikirjoittaa.

```
class Ohjelma extends Applet
{

  public void init()
  {
    // Alusta applet
  }

  public void start()
  {
    // Käynnistä säie
  }

  public void stop()
  {
    // Lopeta säie ja musiikki
  }

 }
```

5.5 Tapahtumafunktioista

Java-ohjelmat ovat tapahtumaohjattuja, kuten yleensä kaikki Windows-ohjelmat. Tapahtumaohjattu tarkoittaa sitä, että käyttäjän suorittamat toimenpiteet, esimerkiksi hiiren napin painallukset, lähetetään ohjelmille viesteillä. Selaimen tehtävänä on reagoida näihin viesteihin,

käsitellä ne ja lähettää eteenpäin. Java-ohjelmissa ei enää nähdä eikä käsitellä varsinaisia viestejä, vaan tapahtumien välitys suoritetaan takaisinkutsuttavillafunktioilla. Yksi esimerkki tällaisesta funktiosta on **`paint`**. Se vastaa ikkunan uudelleen piirtämisestä (parametrina palautetaan Graphics-olio). Paint-funktio voidaan ylikirjoittaa samalla tavalla omaan ohjelmaan kuin edellä esitetyt init-, start- ja stop-funktiot.

```
public void paint (Graphics g)
{
}
```

Paint-funktio on helpoin tapa piirtää grafiikkaa piirtopinnalle. Esimerkiksi täytetyn ympyrän piirtäminen tapahtuu näin:

```
public void paint (Graphics g)
{
 g.fillOval(100, 100, 200, 200);
}
```

Takaisinkutsuttavat funktiot ylikirjoitetaan sisältämään halutut toimenpiteet. Esimerkiksi tapahtumafunktio mouseDown, jota kutsutaan hiiren nappia painettaessa, voidaan ohjelmoida siten, että ohjelman suorittaminen aloitetaan alusta. Lisää tietoa tapahtumafunktioista kerrotaan luvussa *6 Interaktiivisuus*.

Tapahtumafunktioiden nimien kanssa kanssa tulee olla tarkkana, sillä kääntäjä ei tiedä, esitteletkö uuden funktion vai ylikirjoitatko vanhan. Esimerkiksi alla oleva funktio ei reagoi piirtopinnan uudelleen piirtämiseen, koska funktio paint on kirjoitettu väärin (alkaa isolla kirjaimella).

```
public void Paint (Graphics g)
{
 g.fillOval(100, 100, 200, 200);
}
```

5.6 Appletin testaaminen

Kun appletti on kirjoitettu, voidaan aloittaa sen käyttöönotto. Ensimmäinen tehtävä on kääntää lähdekoodi javac-kääntäjällä. Alla on esitetty yksinkertaisen appletin (**TokaApplet**) lähdekoodi, joka on saatu aikaiseksi ylikirjoittamalla init- ja paint-funktiot. Init-funktiossa appletin kooksi asetetaan 400 * 400 pikseliä ja paint-funktiossa tulostetaan ympyrä.

```
import java.awt.*;
import java.applet.*;

public class TokaApplet extends Applet
{
 public void init()
 {
  resize(400,400);
 }

 public void paint (Graphics g)
 {
  g.fillOval(100, 100, 200, 200);
 }
}
```

Valitaan hakemisto, johon esimerkki tallennetaan ja kirjoitetaan sitten komentoriville:

javac TokaApplet.java

HUOM! On tärkeää, että komentoriviargumenttina annetun tiedoston nimi on vastaava kuin luokan nimi - muutoin seuraa virheilmoitus eikä koodia käännetä.

Jollakin ASCII-editorilla kirjoitetaan appletille HTML-tiedosto. Se voi olla esimerkiksi seuraavanlainen:

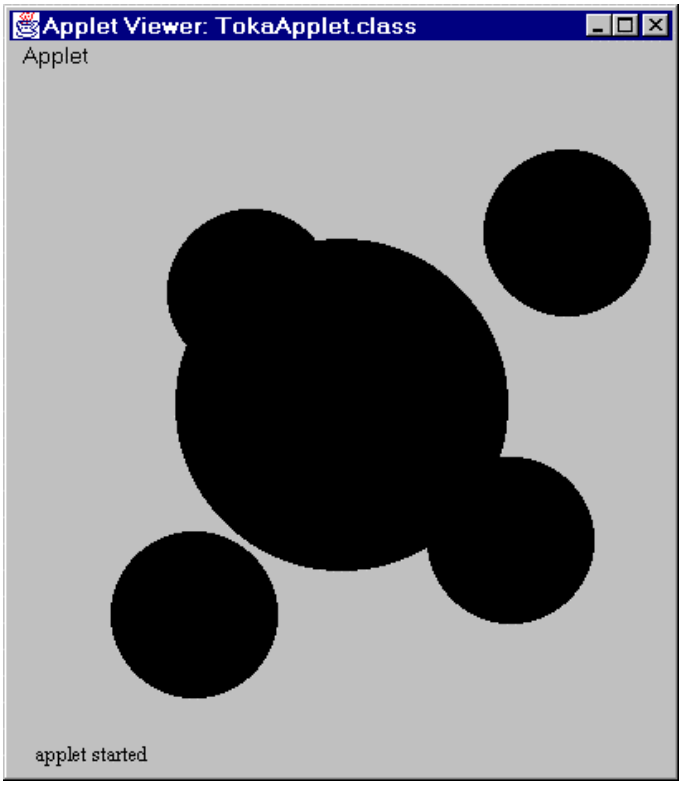
```
<HTML>
<BODY>
<TITLE> TokaApplet </TITLE>
<H1><CENTER> Toinen appletti </CENTER></H1><P>
<APPLET code=tokaapplet.class
   width=400
   height=400>
</APPLET>
</BODY>
</HTML>
```

Kun HTML-tiedosto on tehty, appletin toimintaa voi tutkia millä tahansa Java 1.0-versiota tukevalla selaimella (koska esimerkki ei käytä 1.1-version ominaisuuksia, se toimii moitteetta myös vanhoissa selaimissa). Sellainen on esimerkiksi Netscape Navigator 3.0. Toinen vaihtoehto

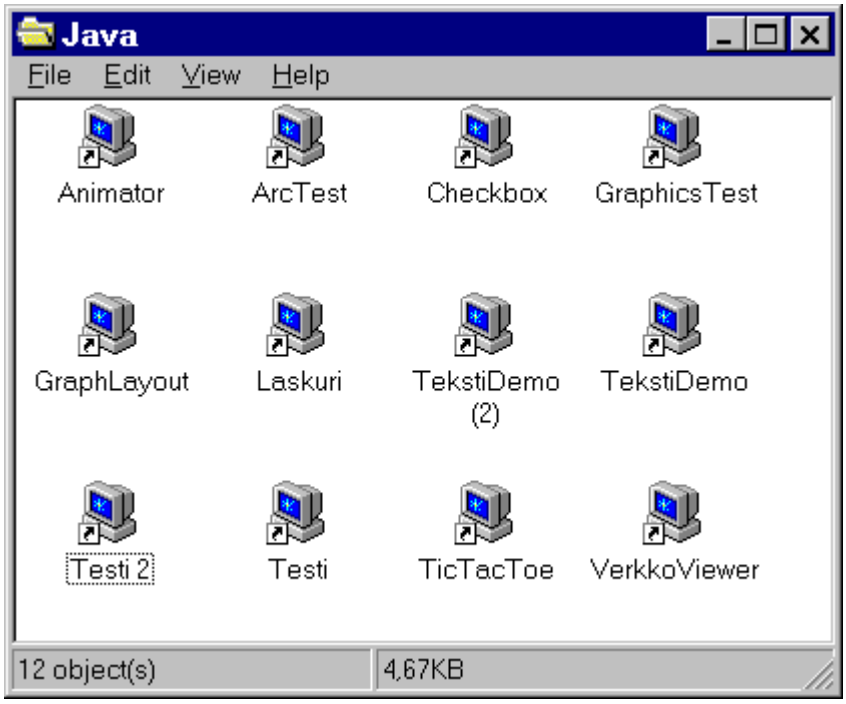
on käyttää Java-ohjelmointipaketin mukana tullutta **appletviewer**-ohjelmaa. Parametrikseen appletviewer tarvitsee HTML-tiedoston, joka sisältää vähintään yhden appletin. HTML-tiedoston sisältöä ei valitettavasti näytetä, joten appletin todellista sijoittumista sivulla ei voi todeta. Appletviewer käynnistetään esimerkiksi komentoriviltä seuraavasti:

appletviewer TokaApplet.html

Kun TokaApplet ajetaan appletviewer-ohjelmalla, saadaan tulokseksi seuraavanlainen ikkuna:



Testusvaiheessa appletviewerin etuna on nopeus. Nopeusero selaimiin johtuu siitä, että selainohjelma on suljettava aina testauksen jälkeen, koska se varastoi appletin koodin muistiin. Jos appletin koodia muutetaan ja valitaan RELOAD-käsky sivun uudelleen päivittämiseksi, niin lopputuloksena on kuitenkin vanha appletti.



Appletviewer on parempi, koska se käynnistyy nopeammin. Jokaiselle ohjelmalle on helppo tehdä kuvake, josta se käynnistyy.

Appletin testaus onnistuu helpoimmin kirjan mukana tulleella editori-ohjelmalla. Editori-ohjelmassa voit kirjoittaa ja muokata lähdekoodia, kääntää lähdekoodin tavukoodiksi, generoida tarvittavan HTML-tiedoston ja käynnistää appletviewer-ohjelman.

5.7 Parametrien lukeminen

Parametrien välittämisestä appletille mainittiin tämän luvun alussa, mutta siitä, kuinka ne ohjelmassa luetaan, ei ole ollut puhetta. Parametrihan annettiin param-tagissa, jossa määriteltiin **parametrin nimi** sekä **parametrin arvo**. Java-ohjelmassa pyydetään tietoa parametrin arvosta sen nimellä. Esimerkiksi, jos jollakin ohjelmalla on seuraavanlainen param-tagi

```
<param name="kuva" value="1">
```

niin parametrin arvo saadaan selville **`getParameter`**-funktiolla, kun funktiolle annetaan parametriksi merkkijono ”kuva”. Tässä tapauksessa funktion paluuarvona on merkkijono yksi ("1"). Mikäli pyydettyä parametria ei ole param-tagilla määritelty, palautetaan määrittelemätön merkkijono.

```
String kuva = getParameter(”kuva”) ;
```

Merkkijono voidaan muuntaa numeeriseksi Integer-olion funktiolla **`valueOf`** ja **`intValue.`** ValueOf palauttaa merkkijonosta Integer-olion j intValue muuttaa Integer-olion int-tietotyypiksi.

```
    int kuva =
        Integer.valueOf(getParameter(”kuva”)).intValue();
```

Lyhyellä ohjelmakoodilla voi testata kuinka parametrien lukeminen tapahtuu.

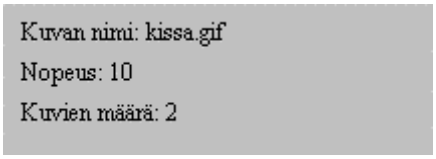
```
import  java.net.URL;
import  java.applet.*;
import  java.awt.*;

public class Parametrit extends Applet
{
  int kuva, nopeus;
  String nimi;

  public void init()
  {
   kuva = Integer.valueOf(getParameter("kuva")).
            intValue();
   nopeus = Integer.valueOf(getParameter("nopeus")).
            intValue();
   nimi = getParameter("nimi");
  }

  public void paint(Graphics g)
  {
   g.drawString("Kuvan nimi: "+nimi,10, 20);
   g.drawString("Nopeus: "+nopeus, 10, 40);
   g.drawString("Kuvien määrä: "+kuva, 10, 60);
  }
}
```

Appletin HTML-tiedosto:

```
<HTML>
<BODY>
<TITLE> Parametrit </TITLE>
<H1><CENTER> Esittelee parametrien lukua </CENTER></H1><P>
<APPLET code=Parametrit.class
   width=100
   height=100>
```

```
<PARAM name="kuva" value="2">
<PARAM name="nopeus" value="10">
<PARAM name="nimi" value="kissa.gif">
</APPLET>
</BODY>
</HTML>
```

Tulos on:



5.8 Appletin URL-osoitteet

Joskus on tärkeää tietää, mitkä ovat HTML-sivun ja appletin URL-osoitteet (voivat olla tietenkin erilaisia). Tällainen tilanne syntyy esimerkiksi silloin, kun halutaan lukea tiedosto, jossa on animaatioesityksen yksi kuva (*image*). Periaatteenahan on, että ohjelmat ovat riippumattomia URL-osoitteista, koska sivu saattaa olla yhtä hyvin omalla kiintolevyllä tai jossakin päin Internetiä. Tiedoston osoittamisen pitää mielellään tapahtua URL-osoitteen ja tiedoston nimellä tai URL-osoitteen, alihakemiston ja tiedoston nimellä. Alihakemistossa voi täten sijaita kaikki animaatioon tarvittavat GIF-tiedostot.
Applet-luokka mahdollistaa tällaisen tiedoston osoittamisen. Kahdella seuraavalla funktiolla voidaan lukea appletin URL-osoite sekä HTML-sivun URL-osoite.

getCodeBase()	Lukee appletin URL-osoitteen
getDocumentBase()	Lukee HTML-sivun URL-osoitteen

Funktioiden palauttama informaatio ei ole pelkkä merkkijono, joka näkyy selaimessa. URL-osoite sisältää tietoa muun muassa protokollasta ja portista. Jotkut applet-luokan funktiot käyttävät hyväkseen näitä tietoja, mutta ihmiselle ne eivät ole kovin selkeää luettavaa. Tarvittaessa URL-osoite voidaan muuttaa selväkieliseksi merkkijonoksi URL-luokan **`toString`**-funktiolla. URL-luokka määritellään tiedostossa **java.net.URL**, joten ylimääräinen import-lause on sisällytettävä lähdekoodiin.

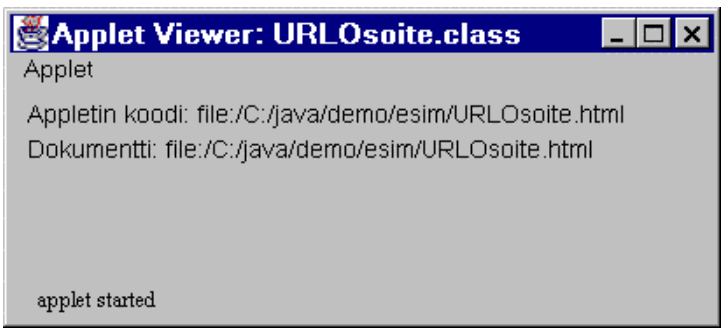
```
import  java.net.URL;
import  java.applet.*;
import  java.awt.*;

public class URLOsoite extends Applet
```

```
{
  String code, doc;

  public void init()
  {
   URL codebase = getCodeBase();
   code = codebase.toString();

   URL docbase = getDocumentBase();
   doc = docbase.toString();
  }

  public void paint(Graphics g)
  {
   g.drawString("Appletin koodi: "+code,10, 20);
   g.drawString("Dokumentti: "+doc, 10, 40);
  }
}
```

Appletin HTML-tiedosto on:

```
<HTML>
<BODY>
<TITLE> URL-osoitte </TITLE>
<H1><CENTER> Esittelee URL-osoitteen lukua </CENTER></H1><P>
<APPLET code=URLOsoite.class
   width=100
   height=100>
</APPLET>
</BODY>
</HTML>
```

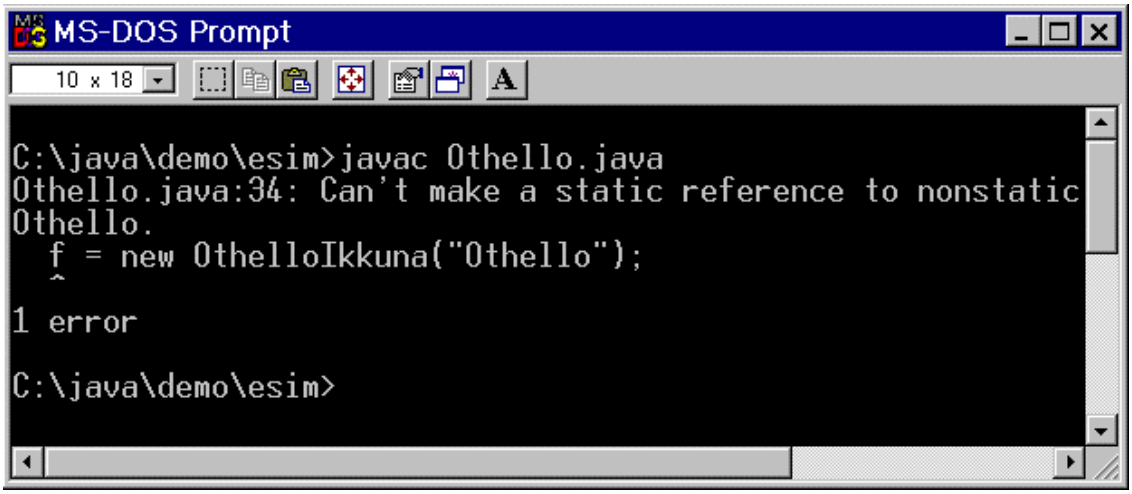
Tulos on:



URL-osoitetta tarvitaan esimerkiksi funktioiden **`play`** ja **`getImage`** yhteydessä. Ensiksi mainitulla soitetaan digitoitua musiikkia ja jälkimmäisellä näytetään GIF-kuvia. Jälkimmäistä käytetään myös animaation tekoon.

5.9 Appletit sovelluksina

Applettia ei voi sellaisenaan ajaa Java-tulkissa, koska siitä puuttuu main-funktio ja main-funktion staattisuus estää selain-ympäristöön tarkoitettujen funktioiden ja jäsenmuuttujien käytön! Sen vuoksi Java-ohjelma, joka on kirjoitettu WWW-ympäristöön, ei toimi sellaisenaan Java-tulkissa, vaikka siihen lisättäisiin puuttuva main-funktio.



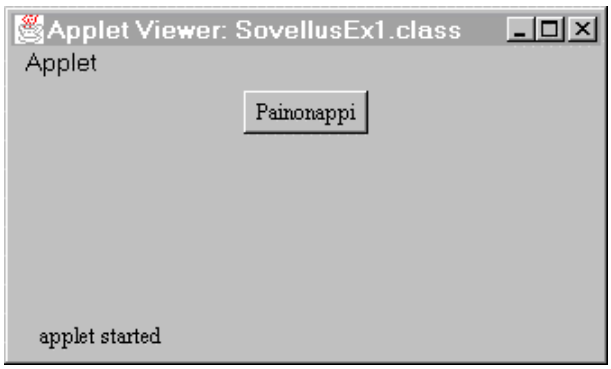

```
public static void main(String args[])
{
  f = new OthelloIkkuna("Otsikko");
}
```

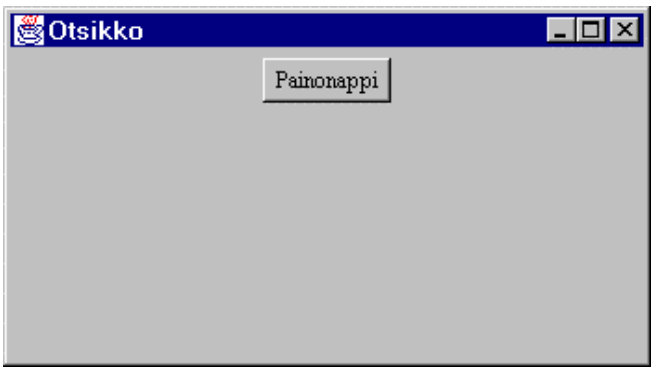
JDK:n mukana tulee luokka **AppletFrame** (*katso myös AppletFrame.java*), joka mahdollistaa main-funktiosta tärkeiden alustusfunktioiden kutsumisen. AppletFrame seuraa ainakin GraphicsTest nimisen demon mukana. Luokkaa hyödynnetään main-funktiossa seuraavasti: kutsutaan funktiota startApplet parametreinaan luokan nimi, otsikko ja parametrit.

```
public static void main(String args[])
{
   AppletFrame.startApplet(String luokka,
                           String otsikko, String params[]);
}
```

Luokan avulla voidaan kirjoittaa ensimmäinen appletti, joka toimii sekä selaimessa että Java-tulkissa. Käännettävä esimerkki on seuraavanlainen:

```
import java.awt.*;
import java.applet.*;
public class SovellusEx1 extends Applet
{
  public void init()
  {
   Button b = new Button("Painonappi");
   add(b);
   resize(300, 300);
  }

  public static void main(String args[])
  {
   AppletFrame.startApplet("SovellusEx1",
    "Otsikko", args);
  }
}
```





Jos halutaan tuottaa toimiva sovellus ilman Appletframe-luokkaa, se onnistuu myös. Esimerkkinä applet nimeltä Ohjelma. Olio tehdään luokasta Ohjelma, jonka init- ja start-funktioita kutsutaan. Sen jälkeen tehdään ikkuna, jonka containeriin liitetään saatu olio. Containerista puhutaan lisää luvussa *7 GUI:n perusteet*.

```
    public static void main(String args[])
    {
      Ohjelma ohjelma = new Ohjelma();
      ohjelma.init();
      ohjelma.start();
```

```
      Frame f = new Frame("Otsikko");
      f.add("Center", ohjelma);
      f.resize(400, 400);
      f.show();
    }
```

Lisäksi tarvitaan tapahtumankäsittelijä, joka tutkii, milloin ikkuna tuhotaan. Tuhoamisen yhteydessä kutsutaan **`System.exit`**-funktiota, joka poistaa sovelluksen. Tapahtumankäsittelijä voi olla alla olevan kaltainen (luokka AppletFrame suorittaa nämä toimenpiteet automaattisesti).

```
    public void windowClosing(WindowEvent e)
    {
      setVisible(false);
      dispose();
      System.exit(0);
    }
```

Lopuksi esitellään vielä käännettävä Java-sovellus, joka voidaan ajaa sekä Java-tulkissa että WWW-selaimessa. Ohjelma on paljon monimutkaisempi kuin edellinen; se johtuu osittain uudesta luokasta, joka joudutaan kirjoittamaan, jotta tapahtumakäsittelijä reagoisi ikkunan sulkemiseen.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class SovellusEx2 extends Applet
{
  public void init()
  {
   Button b = new Button("Painonappi");
   add(b);
   resize(300, 300);
  }

  public static void main(String args[])
  {
   SovellusEx2 sovellus = new SovellusEx2();
   sovellus.init();
   sovellus.start();
   SovellusIkkuna s = new SovellusIkkuna("Sovellus");
   s.add("Center", sovellus);
   s.show();
  }
```

```
}

class SovellusIkkuna extends Frame implements
WindowListener
{
  public SovellusIkkuna(String nimi)
  {
   super(nimi);
   addWindowListener(this);
   setSize(300, 300);
   setVisible(true);
  }

 public void windowClosing(WindowEvent e)
 {
  setVisible(false);
  dispose();
  System.exit(0);
 }

 public  void windowOpened(WindowEvent e){}
 public void windowClosed(WindowEvent e){}
 public void windowIconified(WindowEvent e){}
 public void windowDeiconified(WindowEvent e){}
 public void windowActivated(WindowEvent e){}
 public void windowDeactivated(WindowEvent e){}
}
```

6. Interaktiivisuus

Javan yksi tärkeimmistä ominaisuuksista on kyky käsitellä tapahtumia. Tapahtumilla tarkoitetaan käyttäjän tekemiä toimenpiteitä, esimerkiksi painonapin painamista tai hiiren liikuttamista. Tässä luvussa perehdytään seuraaviin tapahtumiin:

- Elementtien interaktiivisuuteen
- Näppäimistön lukuun
- Hiiren seurantaan

6.1 JDK-versioiden eroja tapahtumankäsittelyssä

Suurimmat muutokset JDK-versioiden 1.0 ja 1.1 välillä ovat tulleet tapahtumankäsittelyyn. Versiossa 1.0 tapahtumankäsittelyn hoitaa aina Component-luokasta periytynyt luokka. Versiossa 1.1 tapahtumienkäsittely perustuu delegointiin - viestit siirretään eteenpäin yhteen tapahtumaluokkaan, jonka ei tarvitse olla Component-luokasta periytynyt.

Tapahtumankäsittely JDK 1.0 -versiossa toimi hyvin pienissä appleteissa, joilla oli yksinkertaisia käyttöliittymiä (vähän komponentteja, luokkia ja koodia). Suuremmat Java-ohjelmat aiheuttavat kuitenkin ongelmia. Esimerkiksi vaatimus alaluokittaa komponentti ennenkuin saadaan mitään toiminnallista käyttöä (interaktiivisuutta) on kömpelöä. Myöskään periyttämismalli ei salli selvän eron luomista ohjelman ja käyttöliittymän välille, koska ohjelmakoodi pitää sijoittaa suoraan komponentin alaluokkaan (ei siis itse ohjelmaan). JDK 1.0 -versiossa kaikki tapahtumat ohjataan aina komponentille riippumatta siitä käsitteleekö komponentti ne vai ei. Esimerkiksi handleEvent-funktiossa voidaan tuntemattoman tapahtuman vuoksi palauttaa väärä palautusarvo. Tällöin ohjelma saattaa toimia arvaamattomasti.

Vaikka JDK 1.1 -versiossa on uusi tapahtumakäsittelytekniikka, sen oppiminen ja täysipainoinen hyödyntäminen ei ole vaikeaa, pikemminkin päinvastoin. Nyt ei ohjelmakoodia tarvitse kirjoittaa komponenttiin, vaan komponentit ja ohjelma pysyvät erillisenä. Vahvuus tässä uudessa tekniikassa on pienentynyt virhealttius ja version1.0 kanssa binääritasoinen yhteensopivuus. Kuitenkaan 1.1-tyyppinen tapahtumankäsittely ei toimi 1.0-versiossa!

HUOM! Koska useat selaimet (Netscape Navigator 3 ja 4, Microsoft Internet Explorer 3) ja sovelluskehittimet (Borland C++ 5.0, Symantec Cafe, Microsoft Visual J++) käyttävät Javan 1.0 -versiota, on yhteensopivuussyistä tähän kirjaan jätetty version 1.0 tapahtumankäsittelystä kertova luku 6.2. Luvusta 6.3 lähtien kerrotaan version 1.1 tapahtumankäsittelystä.

On suositeltavaa että Java-ohjelmat kirjoitetaan uuteen tapahtumankäsittelymalliin. Kahden tapahtumankäsittelymallin sekoittaminen samaan ohjelmaan ei suositeltavaa, mutta ei myöskään

kiellyttyä. Sun on tehnyt parhaimpansa, jotta sekoittaminen onnistuisi. Esimerkiksi Hotjava 1.0 -selain sekoittaa nämä tapahtumankäsittelymallit keskenään, jotta kummankin version appletit toimisivat. AWT tunnistaa (tunnistus lähtee 1.1-versiosta) kumpi tapahtumankäsittelymalli on käytössä ja toimii sen mukaan.

6.2 JDK1.0 - Tapahtumaluokka Event

Tapahtumat JDK 1.0 -versiossa välitetään ohjelmalle takaisinkutsuttavissafunktioissa. Nämä takaisinkutsuttavat funktiot ylikirjoitetaan komponentin alaluokkaan. Yleensä alaluokka on paneeli tai ikkuna. Tapahtumat käsitellään paneelissa, jos tapahtuman aiheuttaja (yleensä graafinen elementti) on asennettu paneeliin add-funktiolla. Tapahtumat käsitellään ikkunassa, jos tapahtuman aiheuttaja on asennettu ikkunaan add-funktiolla.

Esimerkiksi, jos halutaan tutkia hiiren vasempaa nappia, käytetään takaisinkutsuttavaa funktiota `mouseDown`. Tämä funktio ylikirjoitetaan jonkin komponentin alaluokkaan seuraavasti:

```
public class hiiri extends Applet
{
     public void init()
     {
     }

     boolean mouseDown(Event e, int x, int y)
     {
     }
    }
```

Yllä oleva mouseDown on niin sanottu *yksityiskohtainen* tapahtumafunktio. Yksityiskohtaiset tapahtumafunktiot reagoivat johonkin tiettyyn yksinkertaiseen tapahtumaan. Niitä ovat:

mouseEnter	Hiiri saapui komponentin päälle
mouseExit	Hiiri poistui komponentin päältä
mouseMove	Hiirtä siirretty
mouseDown	Hiiren vasen nappi painettu
mouseDrag	Hiirtä liikutetaan nappi painettuna
mouseUp	Hiiren nappi vapautettu
keyDown	Näppäin painettu
keyUp	Näppäin vapautettu
action	Graafinen elementti valittu

Javassa on myös näiden lisäksi *yleinen tapahtumafunktio*, joka reagoi kaikkiin tapahtumiin. Yleistä tapahtumafunktiota tarvitaan, kun tapahtumaa vastaavaa yksityiskohtaista funktiota ei ole olemassa. Yksityiskohtaista funktiota kannattaa käyttää aina kun se on mahdollista ja yleiskäyttöistä silloin, kun vastaavaa yleiskäyttöistä ei ole olemassa.

Yleinen tapahtumafunktio on nimeltään **`handleEvent`**. Tätä tapahtumafunktiota kutsutaan automaattisesti kaikkien tapahtuminen johdosta. HandleEvent-funktio reagoi kaikkiin tapahtumiin ensimmäisenä. Se siis ohittaa kaikki yksityiskohtaiset funktiot. Jos samassa ohjelmassa on kumpiakin, syntyy siitä pieniä ongelmia. Yksi vaihtoehto on kirjoittaa yksityiskohtaisen funktion toiminnot handleEvent-funktion sisään.
HandleEvent-funktio on muotoa:

```
public boolean handleEvent(Event evt)
```

Yleiskäyttöisessä tapahtumankäsittelyfunktiossa ensimmäinen tehtävä on selvittää mistä kyseinen tapahtuma on peräisin. Sen kertoo evt-olion muuttuja **id**. Muuttujan arvoa verrataan evt-oliossa oleviin pysyviin tapahtumamuuttujiin. Niitä ovat muun muassa:

KEY_ACTION	Komennon toteuttava näppäin painettu
KEY_ACTION_RELEASE	Komennon toteuttava näppäin vapautettu
KEY_PRESS	Näppäinpainallus
KEY_RELEASE	Näppäinvapautus
GOT_FOCUS	Komponentilla fokus
LOST_FOCUS	Komponentti menetti fokuksen
LIST_SELECT	Lista valittu
LIST_DESELECT	Listan valinta vapautettu
LOAD_FILE	Tiedoston lataus
SAVE_FILE	Tiedoston tallennus
MOUSE_DOWN	Hiiren vasen nappi painettu
MOUSE_DRAG	Hiirtä vedetään
MOUSE_ENTER	Hiiri saapuu komponentin päälle
MOUSE_EXIT	Hiiri poistuu komponentin päältä
MOUSE_MOVE	Hiirtä siirretty
MOUSE_UP	Hiiren nappi vapautettu
WINDOW_DEICONIFY	Ikkuna suurennettu
WINDOW_DESTROY	Ikkuna tuhotaan
WINDOW_EXPOSE	Ikkuna tuotu näkyviin

WINDOW_ICONIFY	Ikkuna pienennetään kuvakkeeksi
WINDOW_MOVED	Ikkunan sijaintia muutettu
SCROLL_ABSOLUTE	Kohdistin liikkui epämääräisen luvun verran ylöspäin tai alaspäin
SCROLL_LINE_DOWN	Kohdistin liikkui yhden rivin alaspäin
SCROLL_LINE_UP	Kohdistin liikkui yhden rivin ylöspäin
SCROLL_PAGE_DOWN	Kohdistin liikkui sivun alaspäin
SCROLL_PAGE_UP	Kohdistin liikkui sivun ylöspäin

Kun tapahtumatyyppi on selvillä, voidaan evt-olion key-, modifiers-, target- ja arg-muuttujien avulla selvittää tarkka kohde, esimerkiksi näppäin, joka aiheutti tapahtuman. Näiden muuttujien sisällöistä kerrotaan lisää hieman myöhemmin. Jos esimerkiksi halutaan tutkia oliko vapautettu näppäin F3, menetellään näin:

```
public boolean handleEvent(Event evt)
{
 if (evt.id == Event.KEY_RELEASE)
    if (Event.F3 == evt.key)
   // Oli F3
 return true;
}
```

Sekä yleiskäyttöisessä että yksityiskohtaisissa tapahtumafunktioissa esiintyy Event-luokan olio evt. Event-olio on tapahtumien perusolio, joka sisältää tapahtuman kaikki mahdolliset tiedot. Event-olion antamat tiedot ovat erittäin tarpeellisia muun muassa selvittämään tapahtuman aiheuttajan ellei sitä ole selvitetty yksityiskohtaisella funktiolla.

- Hiiren koordinaatit
- Ajan hetki
- Tapahtuman tyyppi
- Tapahtuman aiheuttaja
- Argumentti

Lisäksi Event-olion avulla voi selvittää, mikä on seuraava tapahtuma (jos sellainen on olemassa). Alla on listattu Event-olion muuttujat, joista nämä tiedot selviävät. Tapahtumien eri tyyppeihin, kuten hiiren seurantaan ja näppäimistön lukuun, perehdytään tarkemmin eri lukujen yhteydessä.

id	Tapahtuman tyyppi (esim näppäinpainallus KEY_PRESS)
arg	Argumentti (lisätietoa komponentista)
key	Näppäin, joka aiheutti tapahtuman
modifiers	Erikoisnäppäimet (shift, alt ja ctrl)
target	Tapahtuman aiheuttaja
when	Kellon aika
x	X-koordinaatti tapahtumahetkellä
y	Y-koordinaatti tapahtumahetkellä
evt	Seuraava tapahtuma

Esimerkiksi aika millisekunteina (vrt. Date.getTime()-funktio) saadaan näin:

```
long aika = e.when;
```

tuloksena palautuu tämänhetkinen aika, joka voi olla esimerkiksi

835561200350

Tapahtuman aiheuttaja (target-muuttuja) kertoo millainen komponentti tapahtuman aiheutti. Aiheuttaja voi olla esimerkiksi painonappi (*Button*), jolloin target-muuttuja on Button-luokan olio. Jos painonappeja on useita, tällöin target-muuttuja ei voi yksinään paljastaa oikeaa painonappia. Tämän lisäksi tarvitaan arg-muuttujassa olevaa tietoa. Se kertoisi tässä tapauksessa painonapin nimen. *Argumentin antama tieto vaihtelee tapahtuma kohtaisesti!* Argumenttia tarvitaan enimmäkseen graafisten elementtien yhteydessä kertomaan komponentin nimi tai tila.

```
if ("Napin nimi".equals(arg))
  // oli tämä painonappi
```

Graafisten elementtien tutkimisessa perustana on target-muuttuja. Sen avulla tiedetään, mistä elementistä tapahtuma on peräisin. Tässä käytetään apuna instanceof-lausetta.

```
if (evt.target instanceof Button)
```

Jos yllä oleva if-lause on tosi, käyttäjä on painanut painonappia. Nyt argumentin avulla tutkitaan, mikä oli painonapin nimi. Koska arg-muuttuja on tässä tapuksessa String- olio, käytetään apuna olion equals-funktiota.

```
if ("Painonappi".equals(arg))
```

Tietty tapahtuma voidaan lähettää omalle ohjelmalle ilman käyttäjän avustusta. Ensiksi tehdään Event-olio, jollakin kolmesta muodostimesta. Valinta perustuu siihen kuinka paljon tietoa tarvitaan (riippuu tapahtuman tyypistä). Monimutkaisemmalla saa annettua kaikki tiedot, ja yksinkertaisemmalla voi antaa vain kohteen ja argumentin. Event-luokan muodostimet ovat:

Event(Object target,	Kohde
long when,	Aika
int id,	Tyyppi
int x,	X-koordinaatti
int y,	Y-koordinaatti
int key,	Näppäin
int modifiers,	Erikoisnäppäimet
Object arg)	Argumentti
Event(Object target,	Kohde
long when,	Aika
int id,	Tyyppi
int x,	X-koordinaatti
int y,	Y-koordinaatti
int key,	Näppäin
int modifiers)	Erikoisnäppäimet

Event(Object target,	Kohde
int id,	Tyyppi
Object arg)	Argumentti

Jos halutaan lähettää viesti esimerkiksi painonapille, kannattaa käyttää viimeistä muodostinta. Kohteeksi valitaan Button-luokka, tapahtuman tyypiksi action ja argumentiksi painonapin nimi.

```
Button b;
Event e = new Event(b, Event.ACTION_EVENT, ”Moi”);
```

Kun olio on tehty jotakin muodostinta käyttäen, pitää se lopuksi lähettää eteenpäin (olion luominen ei tarkoita automaattista tapahtuman generointia). Component-luokassa on kaksi erilaista funktiota, joilla tapahtuman voi lähettää. Funktiot ovat:

deliverEvent(Event)	Toimittaa viestin perille
postEvent(Event)	Postittaa viestin

Ensimmäinen funktio palauttaa ohjauksen takaisin ohjelmaan vasta kun viesti on toimitettu perille. Jälkimmäinen funktio postittaa viestin vain eteenpäin.

```
    postEvent(e);
```

Java-sovellus **Postia** lähettää näppäinpainalluksella viestin painonapille ”moro”, että sitä on painettu. Moro vastaa ja kirjoittaa ruudulle tekstin ”Moro sullekin!”.

```
import java.awt.*;
import java.applet.*;

public class Postia extends Applet
{
 Button b;
 public void init()
 {
  setFont(new Font("Arial", Font.BOLD, 28));
  add (new Button("Moro"));
  resize(200,200);
 }

 public boolean action(Event evt, Object arg)
 {
  Graphics g = getGraphics();
  g.drawString("Moro sullekin!",5,180);
  return true;
 }

 public boolean keyDown(Event evt, int key)
 {
  Event e = new Event(b, Event.ACTION_EVENT, "Moro");
  postEvent(e);
  return true;
 }
}
```

6.3 JDK1.1- tapahtumien delegointi

Versiossa 1.1 tapahtumien käsittely on muuttunut melkoisesti, koska tapahtumia ei käsitellä siinä luokassa, johon komponentit on ladattu (ellei erikseen niin määrätä). Tämä johtaa muun muassa siihen, että ikkunassa olevien komponenttien tapahtumienkäsittelyä varten ei tarvitse toteuttaa Frame-luokan alaluokkaa. Tapahtumien käsittelyyn ei myöskään käytetä luvussa 6.2 esiteltyjä takaisinkutsuttaviafunktioita, vaan niihin on omat erityiset funktiot. Funktioista kerrotaan lisää hieman myöhemmin.

6.4 Tapahtumaluokat

Tapahtumia JDK 1.1 -versiossa ei enää edusta yksinään java.awt.Event-luokka numeeristen ID-lukujen avulla, vaan sen sijaan useiden tapahtumaluokkien hierarkia. Jokainen tapahtumaluokka on määritelty siten, että se edustaa tiettyä (pää)tapahtumaa (esim. näppäimistö) ja sen alla olevaa tapahtumajoukkoa (esim. näppäin vapautettu). Tapahtumajoukossa olevat yksittäistapahtumat ovat todellisia käyttäjän aiheuttamia tapahtumia.

Jokaista erityyppistä päätapahtumaa varten tarvitaan Java-ohjelmassa oma luokka. Jos ohjelman pitää hallita esimerkiksi hiiren ja näppäimistön tapahtumat, tarvitaan teoriassa kolme eri luokkaa - yksi pääohjelmalle, toinen hiirelle ja kolmas näppäimistölle. Kuitenkin rajapintojen ominaisuuksien (moniperiyttäminen) ansiosta, yksi luokka voi käsitellä useita erilaisia tapahtumia.

Tapahtumaluokat löytyvät luokkakirjastosta java.util.EventObject. Seuraavassa on lueteltu tapahtumaluokat ja niiden toiminta:

ActionEvent	Tuotetaan, kun käyttäjä valitsee jonkin graafisen elementin.
AdjustmentEvent	Tuotetaan, kun esimerkiksi vierityspalkin kohdistinta siirretään.
ComponentEvent	Tuotetetaan, kun käyttäjä muuttaa komponentin kokoa, sijaintia jne.
ContainerEvent	Tuotetaan, kun containeriin lisätään tai poistetaan komponentteja.
FocusEvent	Tuotetetaan, kun komponentti saa tai menettää fokuksen (käyttäjä aktivoi komponentin).
InputEvent	Päätapahtuma kaikille komponenteista aiheutuneille tapahtumille. Tämä tapahtuma generoidaan ennen varsinaista tapahtumaa.
ItemEvent	Tuotetaan, kun käyttäjä valitsee otsikon (esimerkiksi listasta)

KeyEvent	Tuotetetaan, kun käyttäjä painaa jotakin näppäintä.
MouseEvent	Tuotetetaan, kun käyttäjä painaa hiiren nappia tai liikuttaa hiirtä.
TextEvent	Tuotetetaan, kun käyttäjä muokkaa tekstikomponentin sisältöä.
WindowEvent	Tuotetaan, kun käyttäjä muuttaa ik kunan ominaisuuksia (sulkee, pienentää).

Nämä tapahtumaluokat sisältävät käyttäjän aiheuttaman tapahtuman. Sen voi lukea java.awt.AWTEvent-luokan **`getId`**-funktiolla (kaikki tapahtumaluokat ovat periytyneet AWTEvent-luokasta, joten getId-funktiota voi kutsua tapahtumaolion avulla). Esimerkiksi MouseEvent-tapahtumaoliossa ovat seuraavat tapahtumat:

- MOUSE_CLICKED
- MOUSE_DRAGGED
- MOUSE_ENTERED
- MOUSE_EXITED
- MOUSE_MOVED
- MOUSE_PRESSED
- MOUSE_RELEASED

```
if (me.getId() == MouseEvent. MOUSE_PRESSED)
 // hiiren nappia painettiin
```

Hiiren tapauksessa tapahtumia ei ole välttämätöntä lukea getId-funktiolla, koska voidaan käyttää vastaavia takaisinkutsuttaviafunktiota. GetId-funktiota tarvitaankin enimmäkseen sellaisten tapahtumien kanssa, joille ei ole vastaavaa takaisinkutsuttavaa funktiota tai sitä ei voi käyttää. Tällöin tapahtumienkäsittely suoritetaan aivan eri tavalla. Siitä kerrotaan kuitenkin lisää myöhemmin.

6.5 Tapahtumien sallinta

Tapahtumien jakamisen edellytyksenä on se, että komponentti on sallinut tapahtumat. (Versiossa 1.0 tapahtumien erillistä sallintaa ei tarvitse tehdä.) Automaattisesti, ilman tapahtumien sallintaa, tapahtumia ei lähetetä.

Tapahtumien sallinta tapahtuu **`setListener`**- tai **`addListener`**-funktiolla, riippuen siitä halutaanko yksittäisjaettu vai monijaettu tapahtuma. Esimerkiksi kaikki AWT-komponentit tukevat vain monijaettuja tapahtumia.

- Yksittäisjaettu tapahtuma (`SetListener`-funktio) tarkoittaa sitä, että vain yksi olio voi kuunnella tapahtumia.
- Monijaettu tapahtuma (`AddListener`-funktio) tarkoittaa sitä, että useat oliot voivat kuunnella tapahtumia.

6.6 Kuuntelijat

Tapahtumat Java 1.1 -versiossa käsitellään luokkakirjaston java.util.EventObject **Listener-rajapintojen** tai **Adapter-luokkien** avulla (adaptereista kerrotaan luvussa 6.8). Nämä tavat ovat Sunin suosittelemia. Lisäksi on mahdollista käyttää **process**-alkuisia funktiota, jotka löytyvät komponenteista. Tätä tapaa Sun ei suosittele käyttäväksi, mutta on tilanteita, jossa tätäkin tapaa on pakko käyttää. Process-funktioista kerrotaan luvussa 6.9.

Tapahtumien delegointi tarkoittaa Javassa sitä, että tapahtuma siirretään jonkin luokan käsiteltäväksi. Kaikki luokat eivät sellaisenaan voi tapahtumia käsitellä. Niistä pitää tehdä kuuntelijoita. Kuuntelija-olioiden tehtävänä on tapahtumien odottaminen ja niiden toteuttaminen.

Kuuntelija-olio toteutetaan Listener-rajapinnan avulla. Luokka, joka toteuttaa tämä rajapinnan, voi vastaanottaa ja käsitellä tapahtumia. Listener-rajapinnat määrittelevät yhden tai useamman funktion, jonka avulla kuuntelija käsittelee halutut tapahtumat. Jokaiselle päätapahtumalle on olemassa oma Listener-rajapinta. Ne ovat seuraavat (vertaa rajapintojen nimiä edellä olleisiin tapahtuluokkien nimiin):

- ActionListener
- AdjustmentListener
- ComponentListener
- ContainerListener
- FocusListener
- ItemListener
- KeyListener
- MouseListener
- MouseMoveListener
- TextListener
- WindowListener

HUOM! Koska kuuntelija-olio toteuttaa vain yhden rajapinnan, oliolle ei välitetä koskaan kuin yhden tyyppinen päätapahtuma, esimerkiksi hiiren tapahtumat. (Huomaa, että päätapahtumalla voi olla useita eri tapahtumia. Esimerkiksi hiirellä on tapahtumat mousemove ja mousedown.) Tämä on yksi suuri etu uudessa tapahtumakäsittelyssä - komponenteille ei koskaan lähetetä turhia viestejä, joten prosessoriaikaakaan ei kulu turhaan.

EventListener-rajapinta tyypillisesti omistaa erillisfunktiot jokaiselle tapahtumatyypille, jota päätapahtuma edustaa. Esimerkiksi KeyListener-rajapinta määrittelee funktiot `keyPressed`, `keyReleased` ja `keyTyped`.

HUOM! Kuuntelija-luokan on toteutettava kaikki näistä funktioista, riippumatta siitä mihin tapahtumiin halutaan reagoida.

Esimerkiksi ActionListener-rajapinnassa on vain yksi funktio, joka pitää toteuttaa. Kuuntelija-luokka toteutetaan tällöin seuraavasti:

```
class Kuuntelija implements ActionListener
{
 public void actionPerformed(ActionEvent e)
 {
 }
}
```

Kuuntelija-oliolle syötetään parametriksi tapahtumaolio, joka määrittelee minkälainen tapahtuma on generoitu käyttäjän toimesta. Yllä olevan esimerkkikoodin tapauksessa tapahtumaolio on `ActionEvent`. ActionEvent-oliolla, kuten myös muillakin tapahtumaolioilla, on erilaisia funktioita, joiden avulla saadaan enemmän tietoa tapahtumasta. Esimerkiksi tapahtuman aiheuttaneen komponentin nimen voi lukea **`getActionCommand`**-funktiolla. Kaikilla komponenteilla on vastaavasti **`setActionCommand`**-funktio, jolla tapahtumanimi asetetaan.

```
class Nappaimet implements ActionListener
{
 public void actionPerformed(ActionEvent e)
 {
  if (e.getActionCommand()=="Button")
   // oli Button-niminen komponentti
 }
}
```

Esimerkki ActionListener-rajapinnan käytöstä Button-olion yhteydessä.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Listener extends Applet
{
  public void init()
  {
   Button b = new Button("Button");
   b.addActionListener(new Tapahtuma2(b));
   add(b);
   resize(300, 300);
  }
}

class Tapahtuma2 implements ActionListener
{
 Button b;
 public Tapahtuma2(Button b)
 {
  this.b = b;
 }
 public void actionPerformed(ActionEvent e)
 {
  if (e.getActionCommand()=="Button")
   b.setLabel("MOI");
 }
}
```

6.7 Komponentit kuuntelijoina

Kuuntelija-olio voi olla myös komponentti, joka toteuttaa yhden tai useamman EventListener-rajapinnan. Esimerkiksi Button-olio voi vastata tapahtumiin omassa luokassaan.

Tällä tavoin on helpompi muuttaa 1.0-version Java-ohjelmat 1.1-version kanssa yhteensopiviksi. Seuraavassa on lyhyt esimerkki ActionListener-rajapinnan toteutuksesta Appletissa.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
```

```
public class AwtAction extends Applet implements
ActionListener
{
 Button b;
 public void init()
  {
   b = new Button("Button");
   b.addActionListener(this);
   add(b);
   resize(300, 300);
  }

 public void actionPerformed(ActionEvent e)
 {
  if (e.getActionCommand()=="Button")
   b.setLabel("MOI");
 }
}
```

HUOM! Koska AWT-komponentit tukevat tapahtumien monijaettavuutta, voidaan kuuntelija-olioita lisätä ja poistaa lähteestä missä vaiheessa tahansa. Ei ole kuitenkaan mitään tietoa siitä, missä järjestyksessä kukin kuuntelija-olio saa tapahtuman käsiteltäväkseen. Jos kuuntelija-olio muuttaa tapahtuman ominaisuksuuksia setXXXX-funktiolla, seuraava kuuntelija-olio vastaanottaa muuttuneen tapahtuman, ei alkuperäistä.

6.8 Adapterit

Java-ohjelmassa, joka käyttää AWT:tä, tapahtuman aiheuttaja on tyypillisesti graafinen elementti ja kuuntelija on yleensä sovitin-olio eli **Adapter**. Sovitin-oliolla tapahtumien seuraaminen on käytännössä paljon helpompaa kuin EventListener-rajapinnalla, koska jokaista tapahtumluokan takaisinkutsuttavaa funktiota ei tarvitse ylikirjoittaa. Adapter-luokkien huono puoli on se, että luokkien moniperiytymättömyyden johdosta adapteri-olio edustaa vain tapahtumienkäsittelijää. Sen sijaan EventListener-rajapinnan toteuttaneet luokat voivat edustaa mitä tahansa.

Adapter-luokkia on 6 ja ne ovat seuraavat:

- ComponentAdapter
- FocusAdapter
- KeyAdapter
- MouseAdapter
- MouseMoveAdapter

- WindowAdapter

HUOM! Oletussovitinta ei ole olemassa kaikille tapahtumille.

Adapteri-luokat toteuttavat EventListener-rajapinnan ja sen kaikki funktiot. Esimerkiksi KeyAdapter-luokka toteuttaa KeyListener-rajapinnan ja sen sisältämät funktiot keyPressed, keyReleased ja keyTyped. Tämä tekniikka sallii ohjelmien helposti alaluokittaa Adapter-luokka ja ***ylikirjoittamaan ainoastaan ne tapahtumafunktiot, joista ollaan kiinnostuneita***. Kaikkia tapahtumafunktiota ei tarvitse ylikirjoittaa! Esimerkiksi, jos halutaan reagoida keyPressed-tapahtumaan menetellään seuraavasti:

```
class Nappaimet extends KeyAdapter
{
 public void keyPressed(KeyEvent e)
 {
 }
}
```

Esimerkki KeyAdapter-luokan käytöstä näppäimistön lukemisen yhteydessä. Aktivoimalla painonappi ja painamalla sen jälkeen jotakin näppäintä, vaihdetaan painonapin nimi vastaamaan painetun näppäimen nimeä.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Action extends Applet
{
  public void init()
  {
   Button b = new Button("Button");
   b.addKeyListener(new KeyEventAdapter(b));
   add(b);
   resize(300, 300);
  }
}

class KeyEventAdapter extends KeyAdapter
{
 Button b;
 public KeyEventAdapter(Button b)
```

```
 {
  this.b = b;
 }
 public void keyPressed(KeyEvent e)
 {
  b.setLabel(e.getKeyText(e.getKeyCode()));
 }
}
```

6.9 Tapahtumien prosessointi

Tapahtumien prosessointi tuo Java 1.0 -version piirteitä 1.1-versioon. Prosessoinissa nimittäin ei tarvita lainkaan kuuntelijoita eikä sovittimia. Tällöin tapahtumia ei myöskään siirretä muiden luokkien käsiteltäviksim, vaan tapahtumat käsitellään aina jossakin Component-luokasta periytyneessä luokassa. Yleensä siis ikkunaluokassa tai applettiluokassa.

Tapahtuman prosessointi aloitetaan siten että ensiksi määrätään mitkä päätapahtumat prosessoidaan. Automaattisesti mitään tapahtumia ei prosessoida. Asettaminen tapahtuu funktiolla **`enableEvents`**. Funktio löytyy Component-luokasta eli se periytyy siten myös ikkuna- ja applettiluokkiin. Parametriksi funktiolle annetaan luokasta AWTEvent löytyvä long-tyyppinen arvo. Se määrää siis minkä tyyppiset tapahtumat sallitaan. Arvo on yksi seuraavista.

ACTION_EVENT_MASK	Sallitaan komponentien valinnasta aiheutuneet tapahtumat
ADJUSTMENT_EVENT_MASK	Sallitaan vierityspalkeista tulevat tapahtumat
COMPONENT_EVENT_MASK	Sallitaan komponenteista aiheutuneet tapahtumat
CONTAINER_EVENT_MASK	Sallitaan containerista aiheutuneet tapahtumat
FOCUS_EVENT_MASK	Sallitaan fokuksesta aiheutuvat tapahtumat
ITEM_EVENT_MASK	Sallitaan otsikon valinnasta aiheutuvat tapahtumat
KEY_EVENT_MASK	Sallitaan näppäimistöstä aiheutuvat tapahtumat
MOUSE_EVENT_MASK	Sallitaan hiirestä aiheutuvat tapahtumat
MOUSE_MOTION_EVENT_MASK	Sallitaan hiiren liikkeestä aiheutuvat tapahtumat
TEXT_EVENT_MASK	Sallitaan tekstielementeistä aiheutuvat tapahtumat

WINDOW_EVENT_MASK Sallitaan ikkunoista aiheutuneet tapahtumat

```
enableEvents(AWTEvent.KEY_EVENT_MASK);
```

Kun tapahtumat on sallittu, pitää ylikirjoittaa tapahtumaa vastaava takaisinkutsuttuvatapahtumafunktio. Tapahtumafunktiolle välittyy sama tapahtumaolio kuin kuuntelijoillekin. Periaate tapahtumanjatkokäsittelyyn on samanlaista kaikilla riippumatta siitä millä tavalla tapahtuma on saatu luettua. Tapahtumafunktiot ovat:

processActionEvent(ActionEvent)

processComponentEvent(ComponentEvent)

processContainerEvent(ContainerEvent)

processEvent(AWTEvent)

processFocusEvent(FocusEvent)

processKeyEvent(KeyEvent)

processMouseEvent(MouseEvent)

processMouseMotionEvent(MouseEvent)

processAdjustmentEvent(AdjustmentEvent)

processTextEvent(TextEvent)

processWindowEvent(WindowEvent)

```
public void processKeyEvent(KeyEvent)
```

6.10 Interaktiivisuutta elementeillä

Graafiset elementit ovat GUI:n eli graafisen käyttöliittymän tärkeimmät komponentit. Niistä kerrotaan lisää luvussa 9 *Graafiset elementit*. Tässä luvussa perehdytään graafisten elemettien tapahtumankäsittelyn perusteisiin.

Elementtien tutkimisen perustana on tapahtumaluokka **ActionEvent**.
ActionEvent-olion tärkein funktio on nimeltään **`getActionCommand`**, jolla voi lukea tapahtuman aiheuttaneen elementin nimen (mutta ei esimerkiksi valitun listan otsikon nimeä). Olion funktioita ovat:

getActionCommand() Palauttaa tapahtuman aiheuttaneen elementin nimen.

getModifiers() Palauttaa näppäimet, jotka olivat alhaalla tapahtumahetkellä.

paramString()	Tapahtuman tiedot
getSource()	Palauttaa tapahtuman aiheuttaneen olion.

Tapahtumankäsittely suoritetaan luokassa, joka on toteuttanut ActionListener-rajapinnan. Vastaavaa sovitinluokkaa ei ole olemassa.
Luokan pitää ylikirjoittaa funktio actionPerformed. Funktio on muotoa:

```
public void actionPerformed(ActionEvent e)
```

Tätä tapahtumafunktiota kutsutaan automaattisesti, kun käyttäjä aktivoi elementin. Elementin vapauttamisesta funktio ei kerro mitään. Lisäksi on olemassa elementtejä, jotka käyttävät ActionEvent-tapahtuman lisäksi myös muita tapahtumaolioita, esimerkiksi ItemEvent-oliota. Näistä kerrotaan tarkemmin kyseisten elemettien yhteydessä luvussa 9.

Java-ohjelma **Elementit.java** on lyhyt esimerkki painonapin käyttöönotosta. Ohjelmassa luodaan yksi painonappi nimeltään Button1. Kun nappia painetaan, selaimen tilapalkkiin tulee teksti: ”mua painettiin”.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Elementit extends Applet implements
ActionListener
{
 Button b;
 public void init()
 {
  add (new Button("Button"));
  b.addActionListener(this);
 }

 public void actionPerformed(ActionEvent e)
 {
  if (e.getActionCommand() == "Button")
    showStatus("Mua painettiin");
 }
}
```

6.11 Näppäimistön luku

Näppäimistön lukuun on tarkoitettu tapahtumaluokka nimeltä **KeyEvent**. Luokan funktiot, joilla voidaan lukea tapahtuneen aiheuttaneen näppäimen tiedot, ovat esitetty alla:

getKeyChar()	Palauttaa painetun näppäimen merkin
getKeyCode()	Palauttaa painetun näppäimen koodin
getKeyModifiersText(int)	Palauttaa erikoisnäppäimet merkkijonossa
getKeyText(int)	Palauttaa merkkijonon, jossa näppäimen nimi (esim. F10).
isActionKey()	Tosi, jos näppäin on kytketty johonkin toimintoon

Funktiot, joilla voidaan muokata painetun näppäimen ominaisuuksia, ovat esitetty alla:

setKeyChar(char)	Asettaa painetun näppäimen merkin
setKeyCode(int)	Asettaa painetun näppäimen koodin
setModifiers(int)	Asettaa erikoisnäppäimet

Tapahtumafunktiota **`keyPressed`** kutsutaan automaattisesti, kun käyttäjä painaa jotakin näppäintä. Funktio on muotoa:

```
public void keyPressed(KeyEvent e)
```

Tapahtumafunktiota **`keyReleased`** kutsutaan automaattisesti, kun käyttäjä vapauttaa näppäimen. Funktio on muotoa:

```
public void keyReleased(KeyEvent e)
```

Tapahtumafunktiota **`keyTyped`** kutsutaan automaattisesti, kun käyttäjä painaa jotakin näppäintä ja vapauttaa sen välittömästi. Funktio on muotoa:

```
public void keyTyped(KeyEvent e)
```

Parametrina välitetään KeyEvent-olio, joka sisältää kaikki mahdolliset tiedot tapahtumasta. Edellä esitetyillä funktioilla voi lukea tapahtuneen aiheuttaneen näppäimen koodin, ASCII-merkin ja

näppäimen nimen. Painetun näppäimen tiedot ovat riippuvaisia siitä oliko SHIFT-näppäin painettu vai ei. Jos oli, palautetaan iso kirjain muutoin palautetaan pieni kirjain.

```
s = String.valueOf((char) key);
```

KeyEvent-oliossa on tallennettuna kaikkien näppäinkoodien arvot, jotka voivat tapahtuman aiheuttaa. Vertaamalla niitä `getKeyCode`-funktiolla saatuun arvoon, saadaan selville mitä näppäintä painettiin. Seuraavassa on esitelty joitakin näppäinkoodeja. Muut näppäinkoodit voit katsoa Javan API:n luokasta java.awt.event.KeyEvent.

VK_UP	Nuolinäppäin ylös
VK_DOWN	Nuolinäppäin alas
VK_RIGHT	Nuolinäppäin oikealle
VK_LEFT	Nuolinäppäin vasemmalle
VK_END	End-näppäin
VK_HOME	Home-näppäin
VK_INSERT	Insert-näppäin
VK_DELETE	Delete-näppäin
VK_PGDN	Page Down -näppäin
VK_PGUP	Page Up -näppäin
VK_F1 - VK_F12	Funktionäppäimet F1 - F12
VK_A - VK_Z	Näppäimet A:sta Z:taan

Esimerkiksi funktionäppäimen F3 vertaaminen funktion `getKeyCode` arvoon voidaan toteuttaa näin:

```
if (KeyEvent.VK_F3 == e.getKeyCode)
 // Oli F3
```

Erikoisnäppäinten, kuten shift, alt ja ctrl, lukeminen tapahtuu myös getKeyCode-funktiolla. Kun funktion tulos ohjataan suoraan **`getKeyText`**-funktiolle, palautuu seuraavat merkkijonot, kun jokin erikoisnäppäin on painettu. Vastaavasti palautuu näippäimen nimi, mikäli painettu näppäin ei ollut erikoisnäppäin.

SHIFT	Shift-näppäin
ALT	Alt-näppäin

CTRL Ctrl-näppäin

```
e.getKeyText(e.getKeyCode());
```

Java-ohjelma **Nappain.java** esittelee näppäimistön lukua. Näppäimistön luku on toteutettu keyDown-funktiolla ja näppäimen tulostus paint-funktiolla. Ohjelma tunnistaa normaalit näppäimet, funktionäppäimet ja erikoisnäppäimet. Nappain-ohjelman tulos on seuraava (riippuen tietenkin painetusta näppäimestä):

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Nappain extends Applet implements
KeyListener
{
 Button b;
 String s="";
 public void init()
 {
  add (b = new Button("Valitse minut"));
  setFont(new Font("Arial", Font.BOLD, 28));
  b.addKeyListener(this);
  resize(300, 300);
```

```
  }

  public void keyPressed(KeyEvent e)
  {
   s = e.getKeyText(e.getKeyCode());
   repaint();
  }

  public void keyTyped(KeyEvent e) { }
  public void keyReleased(KeyEvent e) { }

  public void paint(Graphics g)
  {
   g.drawString(s,90,100);
  }
 }
```

6.12 Hiiren seuranta

Hiiren lukuun on tarkoitettu tapahtumaluokat **MouseEvent** ja **MouseMotionEvent**. Luokan funktiot, joilla voidaan lukea tapahtuneen aiheuttaneen näppäimen tiedot, ovat esitetty alla:

getClickCount()	Palauttaa napauksien määrän
getPoint()	Palauttaa hiiren koordinaatit Point-oliossa
getX()	Palauttaa hiiren X-koordinaatin
getY()	Palauttaa hiiren Y-koordinaatin
isPopupTrigger()	Tosi, jos popup-valikon valinta

Javan 1.1-versiossa hiiren käsittely toteutetaan kahdella eri kuuntelija-rajapinnalla. MouseListener-rajapinta vastaa hiiren nappien painalluksista ja vapautuksista. MouseMotionListener-rajapinta vastaa hiiren vetämisestä ja liikkumisesta. Seuraavaksi esitellään MouseListener-rajapinnan funktiot järjestyksessä. Aloitetaan yleisimmästä eli hiiren vasemman napin tutkimisesta.

Hiiren vasemman napin lukuun on tarkoitettu tapahtumafunktio nimeltä **`mousePressed`**. Se reagoi vain painalluksiin, ei vapautuksiin. Funktio on muotoa:

```
public void mousePressed(MouseEvent e)
```

Parametrina välitetään MouseEvent-olio, joka sisältää kaikki mahdolliset tiedot (kuten edellä on esitetty). Olion lisäksi ei välitetä muuta tietoa, vaan ne itse haettava olion funktioiden avulla. Esimerkiksi paikka, jossa hiiren napin painallus tapahtui saadaan luettua näin:

```
public void mousePressed(MouseEvent e)
{
 x = e.getX();
 y = e.getY();
}
```

MouseEvent-olion sisältö on useimmiten välttämätöntä tutkia, koska mitään ylimääräistä tietoa ei välitetä funktion parametreissa. Esimerkiksi, kun tutkitaan ovatko koordinaatit tietyn alueen sisällä, menetellään näin (alue muodostuu koordinaateista 100, 100, 200, 200):

```
public void mousePressed(MouseEvent e)
{
 x = e.getX();
 y = e.getY();
 if (x >= 100 && x < =200 && y >= 100 && y <= 200)
  // hiiren kursorin alueen sisällä
}
```

Komponenttien napautuksiin on tarkoitettu tapahtumafunktio nimeltä **`mouseClicked`**. Se reagoi vain komponenttien painalluksiin, ei muihin painalluksiin tai vapautuksiin. Funktio on muotoa:

```
public void mouseClicked(MouseEvent e)
```

Hiiren vasemman napin lukuun on tarkoitettu tapahtumafunktio nimeltä **`mouseReleased`**. Se reagoi vain napin vapautuksiin, ei painalluksiin. Funktio on muotoa:

```
public void mouseReleased(MouseEvent e)
```

Parametrina välitetään MouseEvent-olio, joka sisältää kaikki mahdolliset tiedot. Olion lisäksi ei välitetä muuta tietoa, vaan ne itse haettava olion funktioiden avulla, kuten edellä on esitetty.

Edellä esitettyjen funktioiden lisäksi on olemassa kaksi käyttökelpoista tapahtumafunktiota, jotka liittyvät Javan graafiseen käyttöliittymään. Nämä funktiot ovat **`mouseEntered`** ja **`mouseExited`**. Edellinen kertoo milloin hiiren kohdistin saapui komponentin päälle ja jälkimmäinen milloin hiiren kohdistin poistui komponentin päältä. Funktiot ovat muotoa:

```
public void mouseEntered(MouseEvent e)
```

Ja

```
public void mouseExited(MouseEvent e)
```

Parametrina välitetään MouseEvent-olio, joka sisältää kaikki mahdolliset tiedot. MouseEvent-olion sisältöä pitää hyödyntää funktioiden avulla, koska koordinaatit jätetään ilmoittamatta.
MouseEnter ja mouseExit -funktioiden yksi tärkeä tehtävä on ilmoittaa, milloin hiiri saapuu ikkunan päälle. Tämä koskee erityisesti tapausta, jolloin käytetään erillistä ikkunaa eli Framea. Esimerkiksi, kun halutaan tietää, milloin hiiren kohdistin saapuu ikkunaan, menetellään näin:

```
public void mouseEntered(MouseEvent e)
{
 saapui = 1;
}
```

Hiiren vetämisen tutkimiseen on tarkoitettu tapahtumafunktio nimeltä **`mouseDragged`**. Vetäminen tarkoittaa sitä, komponentin päällä painetaan hiiren vasen nappi alas ja liikutetaan hiirtä nappi alas painettuna. Funktio sijaitsee MouseMotionListener-rajapinnassa ja se on muotoa:

```
public void mouseDragged(MouseEvent e)
```

Parametrina välitetään MouseEvent-olio, joka sisältää kaikki mahdolliset tiedot. Olio sisältää **x** ja **y** -muuttujat, jotka ilmaisevat hiiren kohdistimen koordinaatit, joissa vetäminen aloitettiin.

Tämän funktion kanssa on hyvä käyttää edellä esiteltyä `mouseReleased`-funktiota. Esimerkiksi, kun halutaan piirtää viiva alkukoordinaateista (x1, y1) loppukoordinaatteihin (x2, y2) menetellään näin:

```
public void mouseDragged(MouseEvent e)
{
 Graphics g = getGraphics();
 x = e.getX();
 y = e.getY();
 g.drawLine(10,10, x, y);
 return true;
}
```

Hiiren liikkumisen havainnointiin on tarkoitettu tapahtumafunktio nimeltä **mouseMoved**. Funktiota kutsutaan, kun hiiri liikkuu ilman vasemman napin painamista. Funktio sijaitsee MouseListener-rajapinnassa ja on muotoa:

```
public void mouseMoved(MouseEvent e)
```

MouseMove-funktion yksi käyttökohde on ilmoittaa graafisissa ohjelmissa piirtokoordinaatit käyttäjälle ja näin helpottaa työskentelyä. Javassa yksi hyvä paikka koordinaateille on tilapalkki selaimen alareunassa. Tilapalkkiin tulostus tapahtuu tutulla **ShowStatus**-funktiolla. Ohjelmakoodissa tulostus toteutetaan esimerkiksi seuraavalla tavalla:

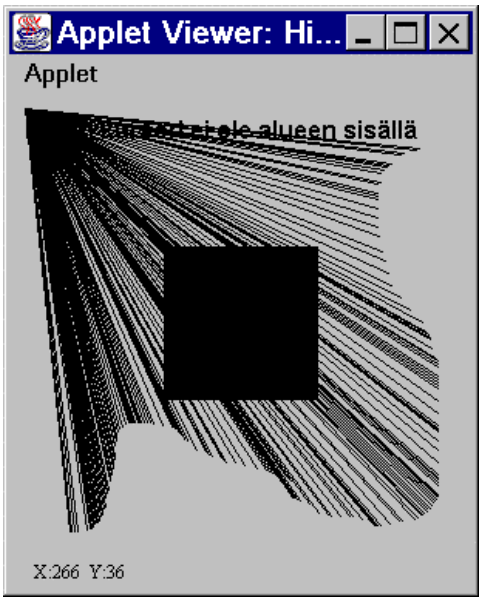
```
public void mouseMoved(MouseEvent e)
{
 x = e.getX();
 y = e.getY();
 showStatus("X:"+x+"  Y:"+y);
}
```

Java-ohjelma **Hiiri.java** esittelee käytännössä hiiren ohjelmointia. Ohjelmassa valjastetaan mouseDown-funktio testamaan onko kohdistin suorakulmaisen alueen sisällä, mouseMove kertoo koordinaatit ja mouseDrag piirtää viivoja.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Hiiri extends Applet implements
MouseListener, MouseMotionListener
{
 String s = "";

 public void init()
 {
  resize(300, 300);
  setFont(new Font("Arial", Font.BOLD, 16));
  addMouseListener(this);
  addMouseMotionListener(this);
 }

 public void mousePressed(MouseEvent e)
 {
  int x = e.getX();
  int y = e.getY();
```

```
   if (x >= 100 && x <= 200 && y >= 100 && y <= 200)
    s = "Hiiren kursori alueen sisällä";
   else
    s = "Hiiren kursori ei ole alueen sisällä";
   repaint();
  }

  public void mouseReleased(MouseEvent e)
  {
   repaint();
  }

  public void mouseDragged(MouseEvent e)
  {
   int x = e.getX();
   int y = e.getY();
   Graphics g = getGraphics();
   g.drawLine(10,10, x, y);
  }

  public void mouseMoved(MouseEvent e)
  {
   int x = e.getX();
   int y = e.getY();
   showStatus("X:"+x+"  Y:"+y);
  }

  public void paint(Graphics g)
  {
   g.drawString(s,10,30);
   g.fillRect(100, 100, 100, 100);
  }

  public void mouseClicked(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}

 }
```

Ohjelman tulos on:


Applet Viewer: Hi...
Applet
X:266 Y:36

7. GUI:n perusteet

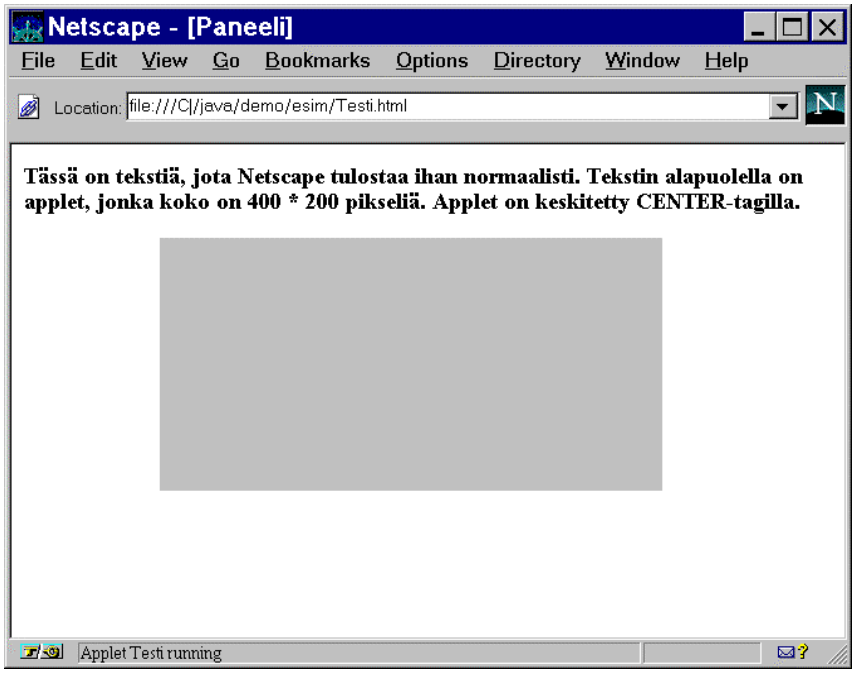
Tämän luvun tarkoituksena on tutustua paneeleihin, ikkunoihin, containereihin sekä komponentteihin. Ne muodostavat yhdessä GUI:n (*Graphical User Interface*) perusteet.

7.1 Paneelit

Paneeli (*panel*) on perusluokka, jonka päällä graafiset toiminnot tapahtuvat. Paneeli on periytynyt Component- ja Container-luokista. Tämän vuoksi paneeleita nimitetään myös containereiksi.

```
+----java.awt.Component
        |
        +----java.awt.Container
              |
              +----java.awt.Panel
```

Paneeli kuvaa selaimen päällä olevaa piirtopintaa. Paneelin koko riippuu appletin näyttöalueen koosta (=appletille varattu tila HTML-sivulta). Viereisessä kuvassa paneeli näkyy harmaalla.

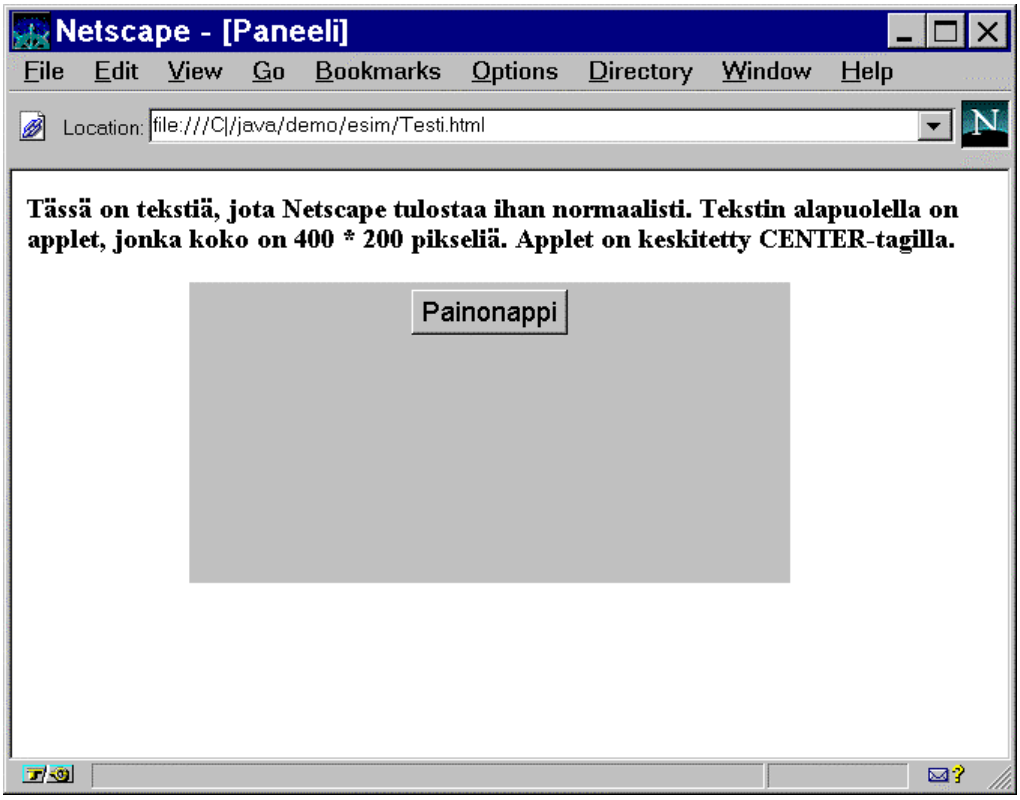


Ohjelmat käyttävät paneelia hyväkseen esittäessään graafisia elementtejä (esimerkiksi painonappeja), grafiikkaa ja kaikkea muuta näkyvää. Näytön osoitteista ja muista konekohtaisista asioista ei tarvitse välittää.

Java-ohjelmat perustuvat Applet-luokkaan, joka on periytynyt paneelista. Tämän vuoksi paneeli on jo valmiiksi avattu ohjelmille, joten grafiikan esittäminen on sinänsä helppoa. Riittää, kun kerrotaan, mitä komponentteja paneelissa halutaan esittää. Esimerkiksi, kun halutaan liittää painonappi, käytetään hyväksi add-funktiota. Esimerkiksi painonapin liittäminen voi tapahtua jo init-funktion aikana.

```
public void init()
{
  add (new Button(”Painonappi”);
}
```

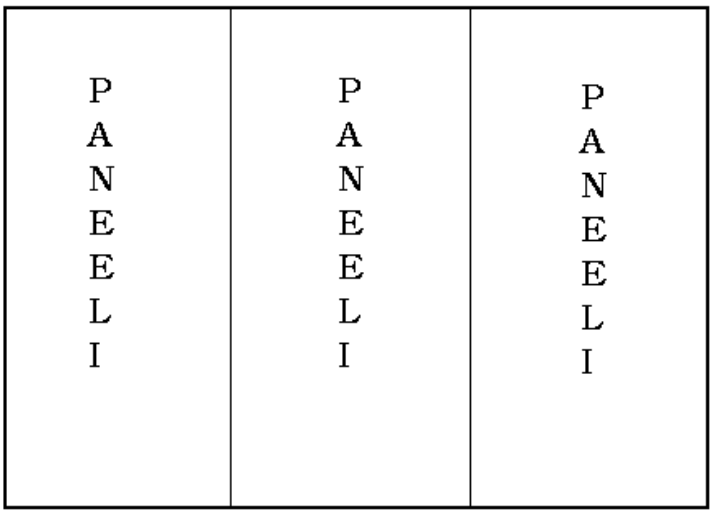
Tuloksena syntyy appletti, joka näyttää selaimessa tällaiselta:



Appletin näyttöalueen koko voidaan tarvittaessa tarkistaa Component-luokan **`getBounds`**-funktiolla. Näyttöalueen koko on ohjelman suorituksen ajan sama, ellei sitä uudelleen muuteta **`setSize`**-funktiolla.

```
public void init()
{
 setSize(300, 300); // näyttöalueen kooksi 300 * 300
}
```

Paneelin koko ei ole välttämättä sama kuin näyttöalueen koko. Se on silloin samankokoinen, kun appletissa on vain yksi paneeli. Jos appletilla on avattu useampia paneeleita, jakaantuu näyttöalueen koko niiden kesken layoutin mukaisesti. Alla olevassa kuvassa appletin näyttöalue on jaettu kolmeen osaan gridlayoutilla. Layouteista puhutaan lisää myöhemmin.



Paneeleja kannattaa käyttää ryhmittämään esimerkiksi graafisia elementtejä, kuten painonappeja ja valintalistoja. Esimerkiksi luvussa *8 Ikkunat, dialogit ja dialogit* tehdään viestilaatikko käyttäen dialogia. Viestilaatikossa dialogi-ikkuna jaetaan pystysuunnassa kahteen osaan siten, että ylemmässä on asiasisältö ja alemmassa painonapit.

Uusi paneeli luodaan Panel-luokan muodostimella. Muodostimia on kaksi kappaletta. Toinen niistä on parametriton ja toinen tarvitsee parametrikseen layoutin. Muodostimen kutsumisen jälkeen palautetaan uusi olio, jonka päälle voidaan kohdistaa graafisia toimintoja.

```
Panel p = new Panel();
```

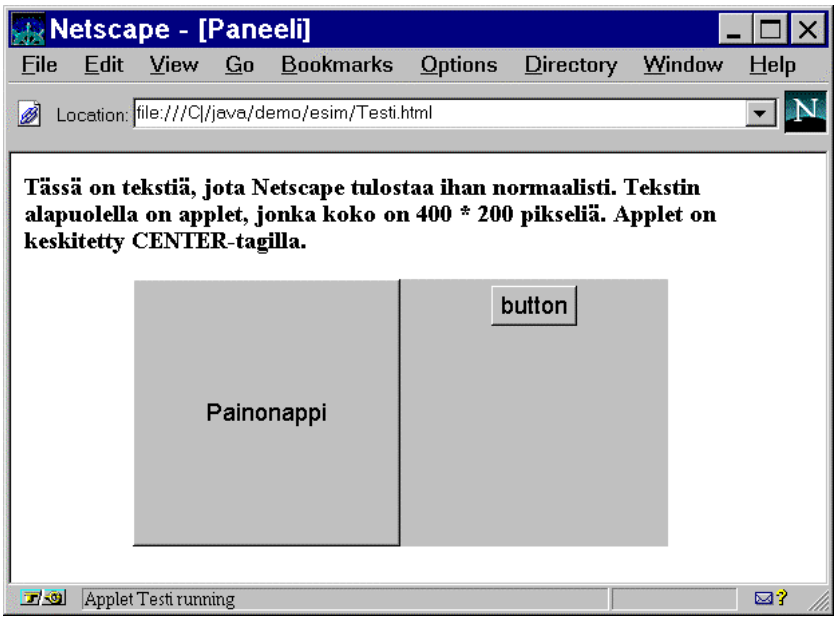
Paneelia ei ole vielä aktivoitu applettiin. Se tehdään kutsumalla Container-luokan add-funktiota. (Panel-luokka itsessään ei sisällä funktiota, jolla se saataisiin aktivoitua.)

```
add (p);
```

Kun paneeli on liitetty, voidaan siihen lisätä esimerkiksi jokin graafinen elementti add-funktiolla. Huomaa, että komennon täytyy kohdistua uuteen paneeliin. Sen vuoksi käytetään uuden p-olion add-käskyä.

```
p.add (new Button(”button”));
```

Nyt painonappi tulee uuteen paneeliin (katso kuva).



Näyttöalueen jako kahteen osaan voidaan toteuttaa esimerkiksi gridlayoutilla. Uuden paneelin layout on flowlayout. Siitä johtuu painonappien erikokoisuus kuvassa.

Uutta paneelia ei tarvitse luoda suoraan tällä muodostimella, vaan se voidaan tehdä oman luokan avulla. Luokka asetetaan periytyneeksi Panel-luokasta.

```
class Paneeli extends Panel
{

 public void paint(Graphics g)
 {
 }

}
```

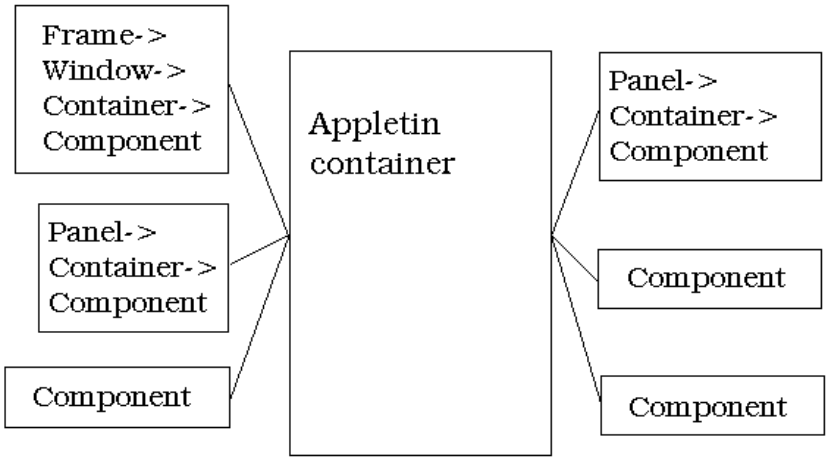
Täten on helpompaa kontrolloida paneelin käyttöä: asettaa komponentteja, tutkia tapahtumia, määrätä layout ja huolehtia muista tärkeistä toimenpiteistä. Esimerkiksi paint-funktio, joka vastaa piirtopinnan uudelleen päivittämisestä, on helppo ylikirjoittaa.

7.2 Container

Container on olio, joka liittyy paneeleihin. Se on paneelin yläpuolella oleva luokka, johon voidaan liittää komponentteja ja uusia containereita. Myös ikkunat (*frame*) sisältävät container-luokan.

```
+----java.awt.Component
        |
        +----java.awt.Container
```

Sana container on suomeksi astia tai säiliö. Suomennos ei ole hyvä, joten tässä käytetään ennemmin nimitystä container. Containerin päätarkoitus on säilöä useita komponentteja itseensä. Komponentteja ovat muun muassa graafiset elementit sekä Component-luokasta periytyneet oliot, kuten paneelit (katso kuva).



Container-oliota ei voi muodostaa yksinään, koska sillä ei ole muodostinta. Se avautuu automaattisesti esimerkiksi paneelien ja ikkunoiden yhteydessä.

Komponentit liitetään container-olioon **`add`**-funktiolla. Kun komponentti on liitetty, se esitetään automaattisesti piirtopinnalla. Komponentti poistetaan **`remove`**-funktiolla. Kun komponentti on poistettu containerista, se poistetaan myös piirtopinnalta. Nämä ovat container-olion tärkeimmät funktiot ja toiminnot.

add (Component)	Lisää komponentin
add (String, Component)	Lisää komponentin ja merkkijonon
remove (Component)	Poistaa komponentin
removeAll ()	Poistaa kaikki komponentit

Seuraavassa on esitelty container-olion muita funktioita.

getComponentCount()	Laskee containerin komponentit
getComponent(int)	Palauttaa komponentin järjestysnumeron perusteella
getComponentAt(int ,int)	Paikallistaa komponentin, joka sijaitsee annetussa pisteessä.
getComponents()	Palauttaa kaikki komponentit
getLayout()	Palauttaa nykyisen layoutin
getInsets()	Palauttaa containerin reunan koon, jolle ei voi kohdistaa graafisia toimenpiteitä
setLayout(LayoutManager)	Asettaa layoutin

7.3 Scrollpane

Javan 1.1-versio helpottaa paneelien ja ikkunoiden vieritystä tuomalla uuden container-luokan nimeltään Scrollpane. Versio 1.0 tukee vierityspalkkeja, mutta niiden käyttäminen ohjelmassa on melko hankalaa. Tämä johtuu osaksi siitä, että vierityspalkit eivät tehneet yhtään mitään: ohjelmoijan piti kirjoittaa koodi, joka reagoi vierityspalkin käyttöön, siirtää vierityspalkin kohdistin oikeaan paikkaan ja lopuksi vielä ohjelmallisesti scrollata näyttöä käyttäjän toiveiden mukaan. Vierityspalkkien sijoittaminenkaan oikeaan paikkaan ei ollut mitään erityisen helppoa. Versiossa 1.1 asiat ovat kuitenkin toisin.
Scrollpane-containeria käytetään kuten tavallisistakin containeria. Ainoa ero on se, että scrollpane ei sisälly automaattisesti mihinkään luokkaan. Scrollpane on siis luotava myöhemmin ja lisättävä johonkin olemassa olevaan containeriin **add**-funktiolla. Kun scrollpane on luotu ja liitetty, siihen voidaan liittää komponentti tai komponentteja. Scrollpane-containerin komponenttien sijoitteluun voidaan käyttää layoutteja.

Scrollpane luodaan ScrollPane-luokan muodostimella. Muodostimia on kaksi kappaletta. Toinen on parametriton ja toinen tarvitsee parametrikseen tiedon siitä, miten vierityspalkkeja sovelletaan.

ScrollPane()	Luo uuden scrollpane-containerin. Vierityspalkit näytetään tarvittaessa.
ScrollPane(int)	Luo uuden scrollpane-containerin. Vierityspalkit näytetään parametrin scrollbarDisplayPolicy mukaan.

Jälkimmäisen muodostimen parametrin arvo on yksi seuraavista:

SCROLLBARS_ALWAYS	Vierityspalkit näytetään aina
SCROLLBARS_AS_NEEDED	Vierityspalkit näytetään tarvittaessa (kun komponentin koko ylittää scrollpane-containerin koon)
SCROLLBARS_NEVER	Vierityspalkkeja ei näytetä koskaan

Muodostimen kutsumisen jälkeen palautetaan uusi olio. Olio pitää liittää olemassa olevaan containeriin ennenkuin scrollpane-containeria voidaan käyttää.

```
ScrollPane sc = new ScrollPane();
add(sc);
```

Kun scrollpane on liitetty, voidaan siihen lisätä esimerkiksi komponentteja tai vaikkapa ylikirjoittaa vain paint-funktio. Tällöin voidaan esittää grafiikkaa, jota voidaan vierittää.

Scrollpane-luokka sisältää käyttökelpoisia funktioita, joilla vieritystä ohjataan. Esimerkiksi ohjelmallisesti vierittäminen rivi kerrallaan tapahtuu funktiolla setScrollPosition, kun x- ja y-parametrien arvot muuttuvat askeleen kerrallaan.

getHAdjustable()	Palauttaa Adjustable-olion, joka kertoo vaakasuoran vierityspalkin tilan.
getHScrollbarHeight()	Palauttaa vaakasuoran vierityspalkin korkeuden
getScrollbarDisplayPolicy()	Palauttaa parametrin scrollbar-DisplayPolicy, jonka mukaan vierityspalkit näytetään.
getScrollPosition()	Palauttaa vierityskohdan nykyisen sijainnin
getVAdjustable()	Palauttaa Adjustable-olion, joka kertoo pystysuoran vierityspalkin tilan.
getViewportSize()	Palauttaa scrollpane-containerin näyttöalueen koon
getVScrollbarWidth()	Palauttaa pystyvierityspalkin korkeuden
setLayout(LayoutManager)	Asettaa layoutin
setScrollPosition(int, int)	Scrollaa haluttuun paikkaan

Vieritysnopeutta säädetään Adjustable-olion funktioilla (olio saadaan luettua getVadjustable- tai getHAdjustable-funktioilla) **`setUnitIncrement`** ja **`setBlockIncrement`**. Oletusarvo yksikkölisäykselle on yksi.

setUnitIncrement(int)	Asettaa yksikkölisäyksen (rivi)
setBlockIncrement(int)	Asettaa blokkilisäyksen (sivu)

```
Adjustable v = sc.getVAdjustable();
Adjustable h = sc.getHAdjustable();
v.setUnitIncrement(10);
v.setUnitIncrement(10);
```

Esimerkki **ImageScroll.java** esittelee scrollapanen käyttöä. Esimerkissä tehdään komponentti, jonka paint-funktiossa piirretään 500 * 500 pikselin kuva. Scrollpanen kooksi annetaan 300 * 300.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class ImageScroll extends Applet
{
```

```
 public void init()
 {
  setSize(320, 320);
  ScrollPane sc = new ScrollPane();
  sc.add (new ImageScroll1());
  sc.setSize(300, 300);

  // säädetään vieritysnopeutta (ei pakollinen)
  Adjustable v = sc.getVAdjustable();
  Adjustable h = sc.getHAdjustable();
  v.setUnitIncrement(10);
  h.setUnitIncrement(10);

  add(sc);
 }

}


class ImageScroll1 extends Component
{
 Image image;

 public ImageScroll1()
 {
  image = Toolkit.getDefaultToolkit()
         .getImage("scrollimage.gif");
 }

 public Dimension getPreferredSize()
 {
  return new Dimension(500, 500);
 }

 public void paint(Graphics g)
 {
  g.drawImage(image, 0, 0, this);
 }
}
```

7.4 Komponentit

Komponentteja ovat painonapit sekä muut graafiset elementit. Komponentteja ovat myös kaikki ne luokat, jotka ovat periytyneet component-luokasta. Esimerkkinä komponentista on Canvas-luokka.

```
+----java.awt.Component
        |
        +----java.awt.Canvas
```

Käytännössä komponentti voi olla melkein mikä tahansa, esimerkiksi ellipsin piirtävä luokka.

```
class Ellipsi extends Component
{
 public Dimension getPreferredSize()
 {
  return new Dimension(200, 200);
 }
 public void paint(Graphics g)
 {
  g.drawOval(0,0,100,150);
 }
}
```

Yllä oleva Ellipsi-luokka piirtää ellipsin, kun se liitetään johonkin Applet-luokan containeriin. Varsinaisen piirtämisen suorittaa paint-funktiossa oleva **`drawOval`**-funktio. Funktiota **`getPreferredSize`** tarvitaan määräämään komponentin koko.

HUOM! Jokaisella itse tehdyllä komponentilla on oltava tietty koko. Se määrätään ylikirjoittamalla **`getPreferredSize`**-funktio ja palauttamalla uusi arvo.

Toimiva applet, joka hyödyntäisi tätä luokkaa, olisi hyvin lyhyt!

```
class PiirraEllipsi extends Applet
{
 public void init()
 {
  add (new Ellipsi());
 }
}
```

Komponentit sijoittuvat näytölle yleensä layoutin sijoittamana, mutta niiden sijaintia voidaan hyvin muuttaa Component-luokan funktioilla. Tällä tavoin komponentit saadaan juuri niihin koordinaatteihin, joihin ne halutaan.

Tehdään ohjelma, joka siirtää edellä olevan ellipsin koordinaatteihin 100,100. Käytetään hyväksi funktiota **`setLocation`**.

```
class PiirraEllipsi extends Applet
{
 Ellipsi e;

 public void init()
 {
  add (e = new Ellipsi());
 }

 public void paint(Graphics g)
 {
  e.setLocation(100,100);
 }
}
```

Muita komponentin sijaintiin vaikuttavia funktioita ovat **`setBounds`** ja **`setSize`**. SetBounds-funktiolla voidaan muuttaa samanaikaisesti komponentin mittoja ja sijaintia. SetSize-funktio puolestaan muuttaa komponentin mittoja ilman että sijainti muuttuu.

getInsets()	Container-luokassa esiintyvä funktio, joka ylikirjoitetaan
setLocation(int,int)	Muuttaa komponentin sijainnin. Parametrit ovat: x-koordinaatti y-koordinaatti
setBounds(int, int, int, int)	Muuttaa komponentin sijainnin ja koon. Parametrit ovat: x-koordinaatti y-koordinaatti suorakulmion leveys suorakulmion korkeus
setSize(int, int)	Muuttaa komponentin koon. Parametrit ovat: suorakulmion leveys suorakulmion korkeus
getSize()	Palauttaa komponentin koon Ei parametreja

Muutetaan edellä olleen paint-funktion sisältöä. Lisätään sinne setBounds-funktio, jolloin voidaan muuttaa komponentin kokoa ja sijaintia.

```
class PiirraEllipsi extends Applet
{
 Ellipsi e;

 public void init()
 {
  add (e = new Ellipsi());
 }

 public void paint(Graphics g)
 {
  e.setBounds(0, 0, 200, 200);
 }
}
```

Tarvittaessa voidaan myös lukea komponentin koordinaatit. Se tapahtuu funktiolla **`getLocation`**. Funktio palauttaa Point-olion, jossa muuttujat x ja y kertovat komponentin koordinaatit.

```
Point p = e.getLocation();
```

Näppärä funktio on myös **`getComponentAt.`** Se palauttaa komponentin, joka sijaitsee annetuissa koordinaateissa.

```
Komponentti = getComponentAt(100,100);
```

Komponentti voidaan asettaa näkymättömäksi tai näkyväksi. Uusi komponentti on lähes aina näkyvissä poislukien Window-luokkaan perustuvat komponentit, kuten ikkuna.

setVisible(false)	Piilottaa komponentin
setVisible(true)	Näyttää komponentin

```
e.setVisible(false);     // piilottaa komponentin
```

Komponentti voi olla myös päällä tai pois päältä.

setEnabled(true)	Sallii komponentin käytön

setEnabled(false)	Estää komponentin käytön

```
e.setEnabled(false);     // Estää komponentin käytön
```

Voidaan myös tutkia missä tilassa komponentti on. Se tapahtuu **is**-alkuisilla funktioilla.

isEnabled()	Tosi, jos komponentin käyttö sallittu
isShowing()	Tosi, jos komponentti näkyvissä ruudulla
isVisible()	Tosi, jos komponentti asetettu näkyväksi

```
if (e.isShowing())
 // komponentti näkyy ruudulla;
```

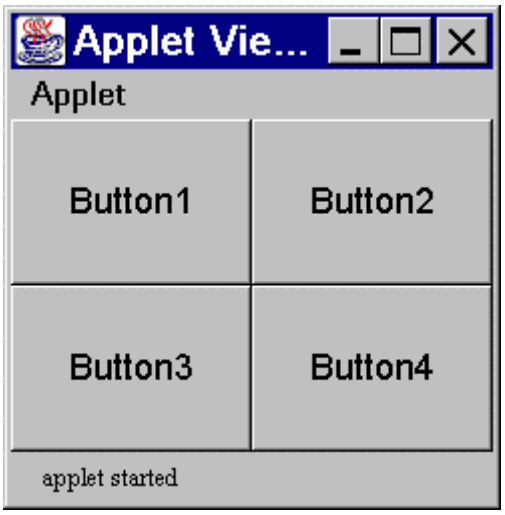
Component-luokassa on paljon mielenkiintoisia funktioita, jotka ovat tarkoitettu graafiseen työskentelyyn. Esimerkkinä mainittakoon taustavärifunktiot **`setBackground`** ja **`getBackground`**, fonttifunktiot **`setFont`** ja **`getFont`**, kohdistimen asetus- ja lukufunktiot **`setCursor`** ja **`getCursor`** sekä piirtopinnan olion lukufunktio **`getGraphics`**. Tärkeät takaisinkutsuttavat funktiot ovat **`paint`** ja **`update`**. Näiden funktioiden käyttöön perehdytään paremmin luvussa *11 Grafiikka*.

7.5 Layoutit

Layoutteja käytetään komponenttien sijoitteluun. Kun containerissa on asennettuna jokin layout, komponentit sijoittuvat tällöin automaattisesti layoutin mukaan. Siten ohjelman lopullinen ulkoasu määräytyy täysin layoutista tyypistä riippuen. Valittavana on viisi erilaista layouttia.

- BorderLayout
- CardLayout
- FlowLayout
- GridLayout
- GridBagLayout

Layout sijoittaa komponentit yleensä liittämisjärjestyksen mukaan tai jonkin muun informaation perusteella. Jokaiselle komponentille varataan tietty tila näytöltä. Esimerkiksi gridlayoutilla voidaan jakaa näyttö neljään yhtäsuureen osaan (katso kuva). Luvun lopussa on lisää asiaa päällekkäisistä layouteista.



Containerilla on oletuksena jokin layout. Jos container on luotu Panel-luokalla, oletuksena on FlowLayout. Jos container on luotu Frame-luokalla, oletuksena on BorderLayout.

Komponenttien välisen etäisyyden paneelin reunoista määrää insets-luokka. Jokaisella paneelilla on tietty oletusarvo. Se voidaan ohittaa ylikirjoittamalla **`getInsets`**-funktio ja palauttamalla uusi arvo.

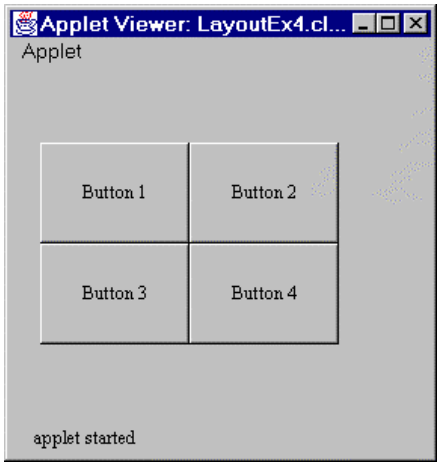
getInsets() Container-luokassa esiintyvä funktio, joka ylikirjoitetaan

Insets-funktion pitää palauttaa uusi Insets-olio. Olio luodaan seuraavalla Insets-luokan muodostimella:

Insets(int, int, int, int) Luo Insets-olion, joka määrää tyhjän tilan komponenttien reunoilla.
Parametrit: yläreuna, vasen reuna, alareuna ja oikea reuna.

```
public class Insets1 extends Applet
{
 public Insets getInsets()
 {
  return new Insets(50, 20, 50, 60);
 }
}
```

Esimerkiksi edellisen kuvan painonapit sijoittuvat nyt viereisen kuvan tavoin. Huomaa, että insets ei vaikuta komponenttien välissä olevaan tilaan. Layoutin avulla on kuitenkin mahdollista vaikuttaa komponenttien väleihin.

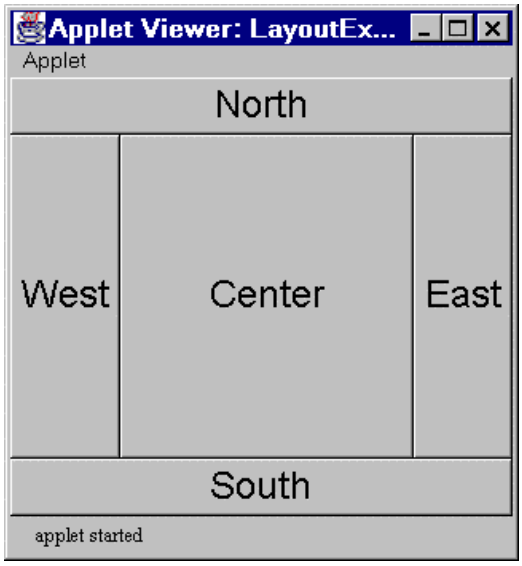


7.6 BorderLayout

BorderLayout eli reuna layout piirtää komponentit näytön reunoille ilmansuuntia apuna käyttäen tai näytön keskelle keskitettäessä. Ilmansuunta tai keskitys kerrotaan container-oliolle komponentin lisäyksen yhteydessä. Mikäli ilmansuuntaa ei anneta, komponentti sijoitetaan oletusarvoisesti keskelle. Ilmansuunnat rajoittavat container-olion komponenttien määrän viiteen, koska useampia komponentteja ei voi tulostaa ilman päällekkäisyyttä. Ilmansuunnat ovat:

North	Pohjoinen
South	Itä
East	Etelä
West	Länsi
Center	Keskellä

Ilmansuunnat "North", "South", "East" ja "West" saavat käyttöönsä tietynlaisen koon näytöltä komponenttia varten. Koon määrää paneelin koko ja muille ilmansuunnille varatut alueet. Keskitetty komponentti saa käyttöönsä kaiken varaamattoman alueen. Jos muita komponentteja ei ole, keskitetty komponentti on koko näytön kokoinen.



BorderLayoutilla on kaksi muodostinta. Ensimmäinen niistä luo oletusasetetun layoutin. Jälkimmäinen luo layoutin, jossa komponenttien välissä on tyhjää tilaa (eli insets-arvo).

BorderLayout()	Oletusasetettu layout
BorderLayout(int, int)	Käyttäjän määrittelemä layout hgap - vaakasuora tyhjä väli vgap - pystysuora tyhjä väli

BorderLayout määritellään new-operaattoria käyttäen ja se liitetään container-olioon setLayout-funktiolla.

```
setLayout (new BorderLayout());
```

Komponentti liitetään containeriin add-funktiolla. Komponentin lisäksi määritellään merkkijono, joka kertoo ilmansuunnan (huomaa isot alkukirjaimet). Jos ilmansuuntaa ei ole, komponentti näytetään keskellä!

```
add ("North", new Button(”Nappi1”));
add ("West", new Button(”Nappi2”));
```

Tämän jälkeen layout sijoittaa annetut komponentit automaattisesti näytölle. Seuraavassa on lyhyt sovellus BorderLayoutista.

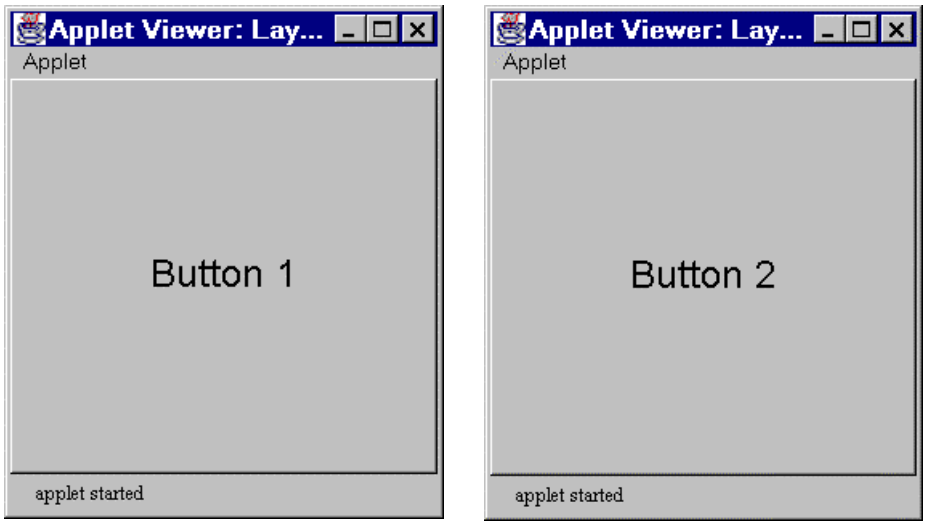
```
import java.awt.*;
import java.applet.*;

public class LayoutEx1 extends Applet
{

 public void init()
 {
  setLayout(new BorderLayout());
  add ("North",new Button("North"));
  add ("South",new Button("South"));
  add ("East", new Button("East"));
  add ("West", new Button("West"));
  add ("Center", new Button("Center"));
 }
}
```

7.7 CardLayout

Cardlayout eli korttilayout on tyyliltään hieman erilainen kuin muut layoutit. Se "kortittaa" näytön siten, että jokainen komponentti saa vuorotellen käyttöönsä koko näytön. Komponentit näytetään

joko järjestyksessä (**`previous`**, **`next`**) tai sekalaisesti (**`first`**, **`last`** ja **`show`**). Container voi omistaa useita komponentteja, mutta vain yksi komponentti on kerrallaan näkyvissä.

Viereisissä kuvissa on käytetty CardLayouttia ja painonappia.
Ensimmäisessä kuvassa on ensimmäinen kortti ja toisessa kuvassa on toinen kortti. Painonapit on nimetty korttien mukaan.



CardLayoutilla on kaksi muodostinta.
Ensimmäinen niistä luo oletusasetetun layoutin. Jälkimmäinen luo layoutin, jossa komponenttien välissä on tyhjää tilaa (insets-arvo).

CardLayout()	Oletusasetettu layout
CardLayout(int, int)	Käyttäjän määrittelemä layout hgap - vaakasuora tyhjä väli vgap - pystysuora tyhjä väli

CardLayout määritellään new-operaattoria käyttäen ja se liitetään container-olioon setLayout-funktiolla.

```
setLayout (new CardLayout());
```

Komponentti liitetään container-olioon add-funktiolla. Add-funktio tarvitsee lisäksi toiseksi parametrikseen kortin nimen.

```
add (new Button(”Nappi1”), ”nappi1”,);
add (new Button(”Nappi2”), ”nappi2”);
```

Tämän jälkeen layout asettaa annetut komponentit automaattisesti koko näytön kokoiseksi ja esittää niistä ensimmäisen komponentin. Muut komponentit saadaan näytölle seuraavien funktioiden avulla.

first(Container)	Näyttää ensimmäisen kortin
last(Container)	Näyttää viimeisen kortin
next(Container)	Näyttää seuraavan kortin

previous(Container)	Näyttää edellisen kortin
show(Container, String)	Näyttää tietyn kortin

Esimerkiksi seuraava kortti saadaan näkyviin näin:

```
    kortti.next(this);
```

Seuraavassa on yksinkertainen Java-ohjelma CardLayoutista.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class LayoutEx2 extends Applet implements
ActionListener
{

 CardLayout kortti;
 Button b;

 public void init()
 {
  setLayout(kortti = new CardLayout());
  add (b = new Button("Button 1"),"kortti1");
  b.addActionListener(this);
  add (b = new Button("Button 2"), "kortti2");
  b.addActionListener(this);
  add (b = new Button("Button 3"), "kortti3");
  b.addActionListener(this);
  add (b = new Button("Button 4"), "kortti4");
  b.addActionListener(this);

 }

 public void actionPerformed(ActionEvent e)
 {
  kortti.next(this);
 }

}
```

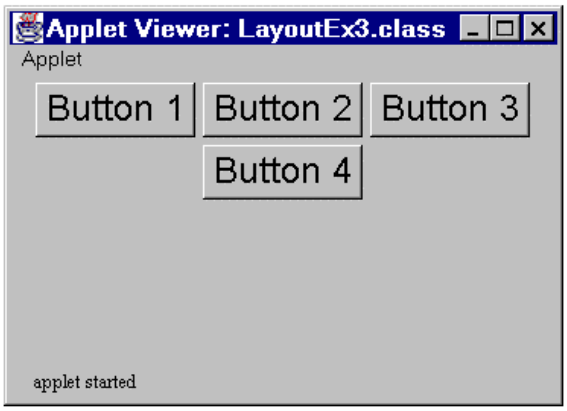
7.8 FlowLayout

FlowLayout eli virta layout on tarkoitettu graafisille elementeille, esimerkiksi painonapeille. FlowLayout sijoittaa komponentit peräkkäiseen järjestykseen, riveittäin vasemmalta oikealle. Sijoitus tapahtuu aina tasauksen mukaan! Mikäli kaikki komponentit eivät mahdu samalle riville, otetaan käyttöön uusi rivi. Näin sijoittamista jatketaan, kunnes kaikki komponentit on sijoitettu.

FlowLayoutin tasauksen voi määrätä tapahtuvaksi keskelle, oikeaan reunaan tai vasempaan reunaan. Tasausliput ovat:

CENTER	Keskelle
LEFT	Vasempaan reunaan
RIGHT	Oikeaan reunaan

Kuvassa on neljä painonappia, jotka on liitetty containeriin. FlowLayout järjestää komponentit automaattisesti riveittäin ja sarakkeittain ja sijoittaa ne näytölle. Tasauslippuna on CENTER.



FlowLayoutilla on kolme muodostinta. Ensimmäinen niistä luo oletusasetetetun layoutin, toinen tasatun layoutin. Viimeinen luo layoutin, jossa on tasaus ja komponenttien välissä tyhjää tilaa.

FlowLayout()	Luo oletusasetetun layoutin
FlowLayout(int)	Luo tasatun layoutin. Parametrina on tasauslippu (CENTER, LEFT tai RIGHT).
FlowLayout(int, int, int)	Luo käyttäjän määrittelemän layoutin, jossa parametrit ovat: flag - Tasauslippu (esitetty yllä) hgap - Vaakasuora tyhjä väli vgap - Pystysuora tyhjä väli

FlowLayout määritellään new-operaattoria käyttäen ja se liitetään container-olioon setLayout-funktiolla.

```
setLayout (new FlowLayout());
```

Komponentti liitetään container-olioon add-funktiolla

```
    add (new Button(”Nappi1”));
    add (new Button(”nappi2”));
```

Tämän jälkeen layout sijoittaa annetut komponentit automaattisesti näytöllä peräkkäiseen järjestykseen. Seuraavassa on yksinkertainen Java-ohjelma FlowLayoutista.

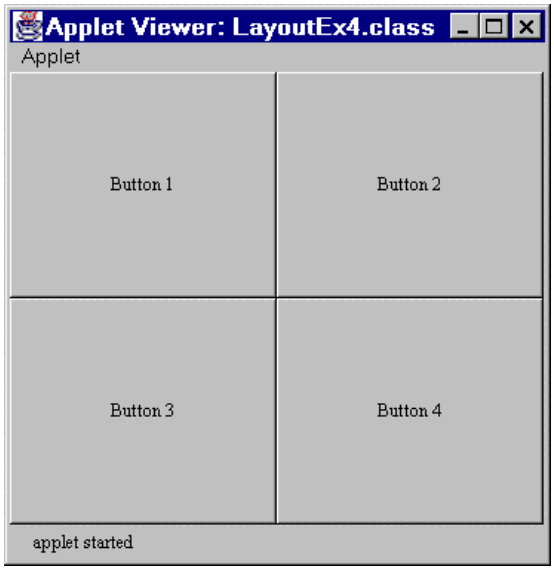
```
import java.awt.*;
import java.applet.*;

public class LayoutEx3 extends Applet
{

  public void init()
  {
   setLayout(new FlowLayout());
   add (new Button("Button 1"));
   add (new Button("Button 2"));
   add (new Button("Button 3"));
   add (new Button("Button 4"));
  }
}
```

7.9 GridLayout

GridLayout eli verkko layout jakaa näytön pinta-alaltaan samankokoisiin soluihin (=palasiin). Solujen koko määräytyy solujen lukumäärästä ja paneelin koosta. Paneeli jaetaan soluihin siten, että määrätään kuinka monen osaan paneeli jaetaan vaakasuunnassa (rivit) ja pystysuunnassa (sarakkeet).

solujen lukumäärä = rivit * sarakkeet.

Komponentit, jotka lisätään containeriin, sijoitetaan soluihin vasemmalta oikealle ja ylhäältä alaspäin. Yhdessä solussa voi olla yksi komponentti, joten enimmillään komponentteja voi olla yhtä paljon kuin on solujakin.



Kuvassa on neljä painonappia, jotka on liitetty containeriin. GridLayout järjestää komponentit automaattisesti soluihin, vasemmalta oikealle ja ylhäältä alas. Komponentit täyttävät solun kokonaan.

GridLayoutilla on kaksi muodostinta. Ensimmäinen niistä luo layoutin, jossa näyttö jaetaan yhtäsuuriin, toisissaan kiinni oleviin soluihin. Jälkimmäinen muodostin luo layoutin, jossa näyttö jaetaan yhtäsuuriin soluihin, mutta jokaisen solun välissä on tyhjää tilaa.

HUOM! Muodostimien rows ja cols -parametrit eivät saa olla yhtäaikaa nollia! Mikäli parametrien arvo on sopimaton, generoidaan IllegalArgumentException-poikkeustilanne.

GridLayout(int, int) Luo GridLayotin
rows - Rivien lukumäärä
cols - Sarakkeiden lukumäärä

GridLayout(int, int, int, int) Luo GridLayotin, jossa solujen välissä tyhjää tilaa.
rows - Rivien lukumäärä
cols - Sarakkeiden lukumäärä hgap - Vaakasuora tyhjä väli vgap - Pystysuora tyhjä väli

GridLayout määritellään new-operaattoria käyttäen ja se liitetään container-olioon setLayout-funktiolla.

```
setLayout (new GridLayout(2,2));
```

Komponentti liitetään container-olioon add-funktiolla

```
add (new Button(”Terve”));
add (new Button(”Moro”));
```

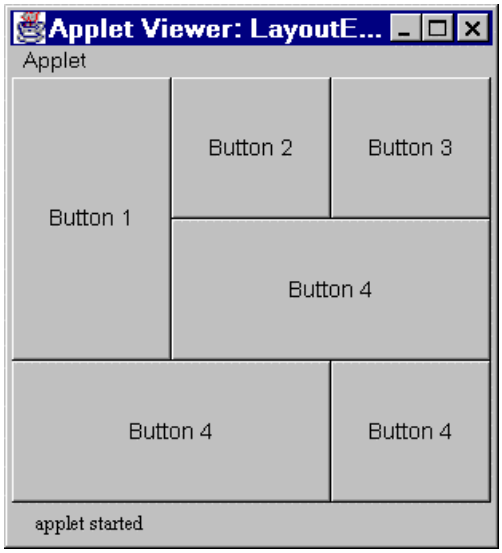
Tämän jälkeen layout sijoittaa annetut komponentit automaattisesti oikeisiin soluihin. Seuraavassa on yksinkertainen Java-ohjelma GridLayoutista.

```
import java.awt.*;
import java.applet.*;

public class LayoutEx4 extends Applet
{

 public void init()
 {
  setLayout(new GridLayout(2,2));
  add (new Button("Button 1"));
  add (new Button("Button 2"));
  add (new Button("Button 3"));
  add (new Button("Button 4"));
 }
}
```

7.10 GridBagLayout

GridBagLayout on myös verkko layout, mutta paljon joustavampi kuin GridLayout. GridBagissa ei tarvitse jakaa näyttöä samankokoisiin soluihin, vaan solut voivat olla minkä kokoisia tahansa. Edelleen yhdessä solussa esiintyy yksi komponentti ja komponentit sijoitetaan soluihin vasemmalta oikealle ja ylhäältä alaspäin, eli samassa järjestyksessä kuin GridLayoutissa.

GridBagLayout käyttää suorakulmaisia soluja. Jokainen komponentti, joka on liitetty GridBagLayouttiin, on yhteydessä GridBagConstraints-olioon. Olion tarkoituksena on kertoa, kuinka näyttö jaetaan osiin ja miten komponentti sijoitetaan näytölle. Sijoitus määrätään 12:lla eri muuttujalla. Muuttujista tärkeimmät ovat **gridwidth**, **gridheight** ja **fill**. Komponenttien tilanvaraus määräytyy lopullisesti, kun koko piirtoalue on täyttynyt komponenteista.



GridBagLayoutilla on vain yksi muodostin. Se luo oletusasetetun layoutin.

GridBagLayout() Luo oletusasetetun GridLayotin

GridBagLayout määritellään new-operaattoria käyttäen ja se liitetään container-olioon setLayout-funktiolla. Koska gridbaglayout-oliota tarvitaan GridBagConstraints-oliossa, kannattaa olio ”ottaa talteen”.

```
GridBagLayout gridbag = new GridBagLayout();
setLayout (gridbag);
```

Kun layout on luotu, on seuraavaksi tehtävä GridBagConstraints-olio, jotta voitaisiin liittää komponentteja. Alla on listattu GridBagConstraints-olion 12 eri muuttujaa, jotka vaikuttavat komponentin sijoitukseen.

gridx Komponentin X-sijainti piirtoalueella.
RELATIVE - komponentti sijoitetaan edellisen komponentin oikealle puolelle (oletus).

gridy Komponentin Y-sijainti piirtoalueella.
RELATIVE - komponentti sijoitetaan edellisen komponentin alapuolelle (oletus).

gridwidth Komponentin leveys soluissa. Oletus 1.
REMAINDER - viimeinen komponentti rivillä
RELATIVE - toiseksi viimeinen komponentti rivillä

gridheight Komponentin korkeus soluissa. Oletus 1.
REMAINDER - viimeinen komponentti rivillä
RELATIVE - toiseksi viimeinen komponentti rivillä

fill Täyttää komponentin täysikokoiseksi.
NONE - ei täyttöä (oletus)
HORIZONTAL-täyttää vaakasuunnassa
VERTICAL - täyttää pystysuunnassa
BOTH - täyttää kummassakin suunnassa

ipadx Lisää komponentin leveyttä. Komponentin leveys on vähintään minimikoko + ipadx * 2.

ipady Lisää komponentin korkeutta. Komponentin korkeus on vähintään minimikoko + ipady * 2.

insets Määrittelee pienimmän välin, joka on komponentin ja piirtoalueen reunojen välillä.

anchor Käytetään sijoittamaan komponentti, kun näyttöalue on suurempi kuin komponentti.
CENTER keskellä (oletus)
NORTH pohjoinen

NORTHEAST	koillinen
EAST	itä
SOUTHEAST	kaakko
SOUTH	etelä
SOUTHWEST	lounas
WEST	länsi
NORTHWEST	luode

weightx Käytetään määräämään tilan jako komponentin ja piirtoalueen reunan välissä. Vähintään yhdellä komponentilla rivillä kannattaa olla painoarvo, muuten kaikki komponentit keskittyvät!

weighty Käytetään määräämään tilan jako komponentin ja piirtoalueen reunan välissä. Vähintään yhdellä komponentilla sarakkeessa kannattaa olla painoarvo, muuten kaikki komponentit keskittyvät!

GridBagConstraints-olio luodaan new-operaattoria käyttäen. Kutsutaan muodostinta parametrittomasti:

```
GridBagConstraints c = new GridBagConstraints();
```

Tämän jälkeen halutut ominaisuudet kirjoitetaan GridBagConstraints-olioon. Luodaan myös komponentti, jota käytetään layoutissa.

```
c.fill = GridBagConstraints.BOTH;
c.weightx = 1.0;
button = new Button (”Painonappi”);
```

Lopuksi kutsutaan funktiota **`setConstraints`**, joka asettaa nämä tiedot layouttiin.

```
gridbag.setConstraints(button, c);
```

Lopuksi komponentti lisätään containeriin add-funktiolla.

```
add (button);
```

Seuraava esimerkki esittelee GridBaglayoutin toimintaa. Siinä luodaan 8 eri painonappia, jotka sijoitetaan erikokoisina eri paikkoihin.

```
import java.awt.*;
import java.applet.*;

public class layoutEx5 extends Applet
{
 Button button;

 public void init()
 {
  GridBagLayout gridbag = new GridBagLayout();
  GridBagConstraints c = new GridBagConstraints();
  //setFont(new Font("Helvetica",Font.PLAIN,14));
  setLayout(gridbag);

  button = new Button("Button 1");
  c.fill = GridBagConstraints.BOTH;
  c.weightx = 1.0;
  c.weighty = 1.0;
  c.gridheight = 2;  // Korkeus 2 elementtiä
  gridbag.setConstraints(button, c);
  add(button);

  button = new Button("Button 2");
  c.gridheight = 1;
  gridbag.setConstraints(button, c);
  add(button);

  button = new Button("Button 3");
  //viimeinen rivillä
  c.gridwidth = GridBagConstraints.REMAINDER;
  gridbag.setConstraints(button, c);
  add(button);

  button = new Button("Button 4");
  //viimeinen rivillä
  c.gridwidth = GridBagConstraints.REMAINDER;
  gridbag.setConstraints(button, c);
  add(button);

  button = new Button("Button 4");
  c.gridwidth = 2;  // leveys 2 elementtiä
  gridbag.setConstraints(button, c);
  add(button);

  button = new Button("Button 4");
  //viimeinen rivillä
```

```
  c.gridwidth = GridBagConstraints.REMAINDER;
  gridbag.setConstraints(button, c);
  add(button);
 }
}
```

7.11 Layouttien yhdistäminen

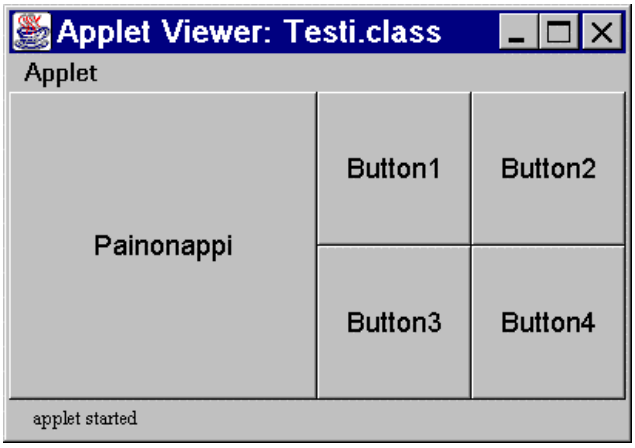
Appletin container-olio voi sisältää toisen container-olion, jolloin myös tähän containeriin voidaan määritellä uusi layout. Esimerkiksi, jos containeriin lisätään komponentti ja uusi paneeli, paneelin voidaan liittää painonappeja sekä layout. Näin komponentit järjestäytyvät paneeleissa omien layouttien mukaisesti. Uusi paneeli liitetään containeriin add-funktiolla.

```
// tee uusi paneeli ja liitä
Panel p = new Panel();
add (p);
```

Paneelin layout asennetaan tutulla setLayout-funktiolla. Huomaa, että se kohdistetaan p-olioon. Samoin myös komponentit liitetään p-olioon.

```
// aseta uusi layout ja komponentit
p.setLayout(new GridLayout(2,2));
p.add(new Button("Button1"));
p.add(new Button("Button2"));
```

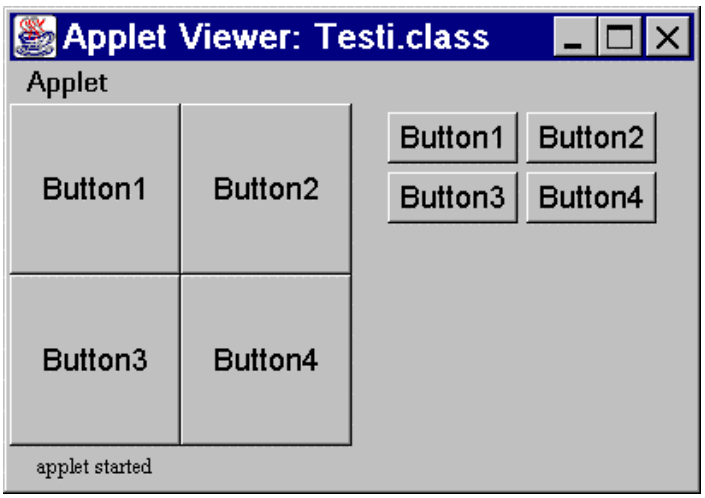
Kuvassa on kaksitasoinen layout. Containerin layouttina toimii gridlayout jakaen näytön kahteen osaan. Toisessa osassa on painonappi, toisessa on paneeli. Paneelin layouttina on myös gridlayout, joka jakaa paneelin neljään osaan.



Ohjelmakoodi näyttää tällaiselta:

```
import  java.applet.*;
import  java.awt.*;

public class KaksiLayout extends Applet
{

  public void init()
  {
   // aseta layout ja painonappi
   setLayout(new GridLayout(1,2));
   add (new Button("Painonappi"));

   // tee uusi paneeli, ja liitä containeriin
   Panel p = new Panel();
   add (p);

   // aseta uusi layout ja komponentit
   p.setLayout(new GridLayout(2,2));
   p.add(new Button("Button1"));
   p.add(new Button("Button2"));
   p.add(new Button("Button3"));
   p.add(new Button("Button4"));
  }
 }
```

Alla olevassa kuvassa on kolmitasoinen layout. Appletin containerin layoutiksi on asennettu gridlayout. Se jakaa näytön kahteen osaan, joista kumpaankin on laitettu paneeli. Paneelien layoutteina toimivat gridlayout ja flowlayout.



Ohjelmakoodi näyttää tällaiselta:

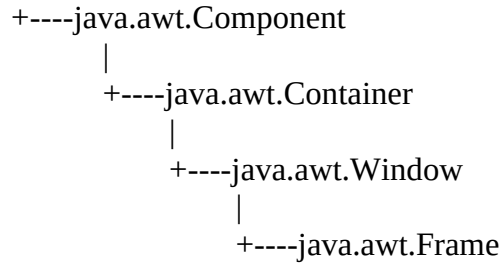
```
import  java.applet.*;
import  java.awt.*;

public class KolmiLayout extends Applet
{

  public void init()
  {
   // Aseta containerin layout
   setLayout(new GridLayout(1,2));

   // Tee uusi paneeli, ja lisää se containeriin
   Panel p = new Panel();
   add (p);

   // Aseta paneelin layout ja siihen komponentit
   p.setLayout(new GridLayout(2,2));
   p.add(new Button("Button1"));
   p.add(new Button("Button2"));
   p.add(new Button("Button3"));
   p.add(new Button("Button4"));

   // Tee toinen paneeli, ja lisää se containeriin
   Panel a = new Panel();
   a.setLayout(new FlowLayout());
   add (a);
   a.add(new Button("Button1"));
   a.add(new Button("Button2"));
   a.add(new Button("Button3"));
   a.add(new Button("Button4"));
  }
}
```

8. Ikkunat, valikot ja dialogit

Tässä luvussa tutustutaan ikkunoihin, valikoihin ja dialogeihin. Ne tuovat webbiin uusia ulottuvaisuuksia. Ikkunan käyttöä kannattaa harkita tarkkaan, sillä se poikkeaa melko rajusti totutusta WWW-kulttuurista. Yleensä kaikki esitykset on totuttu näkemään selaimen päällä.

8.1 Ikkunat

Ikkunalla tarkoitetaan selaimesta erillistä, itsenäistä ikkunaa, jolla on vähintään nimipalkki (*titlebar*) ja piirtoalue (*client area*). Ikkunoissa voi olla myös valikko, josta käyttäjä voi valita valikkokomentoja.



Ikkuna muodostetaan luokalla **Frame**. Luokassa on kaksi muodostinta, jotka ovat parametreiltaan yksinkertaiset: toinen on muodostimista parametriton ja toinen tarvitsee ikkunan nimen.

Frame()	Luo ikkunan
Frame(String)	Luo nimellisen ikkunan

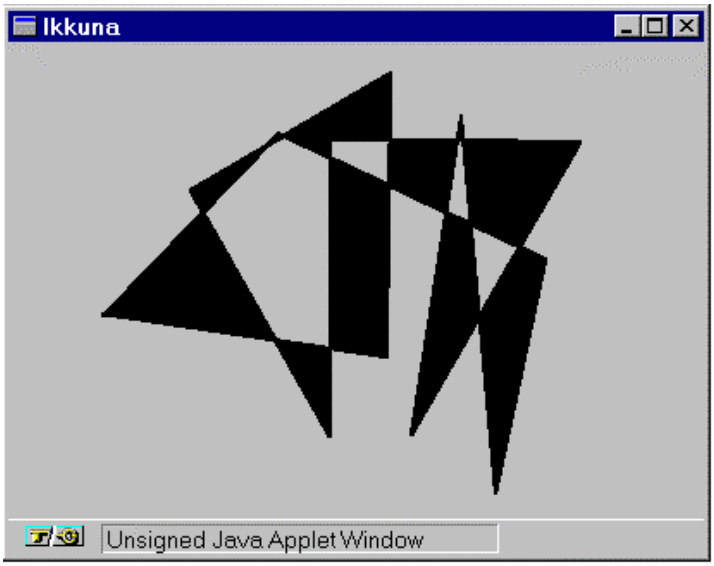
Ikkuna voidaan luoda esimerkiksi näin:

```
Frame ikkuna = new Frame(”Ikkuna”);
```

Kun muodostinta on kutsuttu, järjestelmä luo ikkunan, joka on piilotettu. Sen esiin tuomiseksi kutsutaan `setVisible`-funktiota parametrinaan `true`. Kutsumalla **`show`**-funktiota ikkuna tuodaan päälimmäiseksi, mikäli ikkuna on tuotu näkyviin setVisible-funktiolla.

```
ikkuna.setVisible(true);
```

Nyt ikkuna avautuu kuvaruudulle. Ikkuna on aivan kuin mikä tahansa muukin käyttöjärjestelmän ikkuna, paitsi että sillä on alareunassa tilapalkki, jossa on varoitusteksti (varoitusteksti esiintyy vain appleteilla). Teksti varoittaa käyttäjää siitä, että ikkunan on luonut appletti eikä selain. Näin ollen siihen ei kannata kirjoittaa kovin arkaa tietoa, kuten salasanoja yms. Varoitustekstiä ei voi mitenkään poistaa. Varoitustekstin sanoma riippuu myös selaimesta, alla olevassa kuvassa on Netscape 3.0 -selaimen tuottama ikkuna.



Uuden ikkunan avaaminen kannattaa toteuttaa siten, että tehdään oma luokka, joka periytyy luokasta Frame. Luokan nimeksi annetaan esimerkiksi Ikkuna.

```
class Ikkuna extends Frame
```

Sen jälkeen omaan luokkaan kirjoitetaan muodostin, joka suorittaa tarvittavat toimenpiteet: nimeää ikkunan, tuo ikkunan esiin jne.

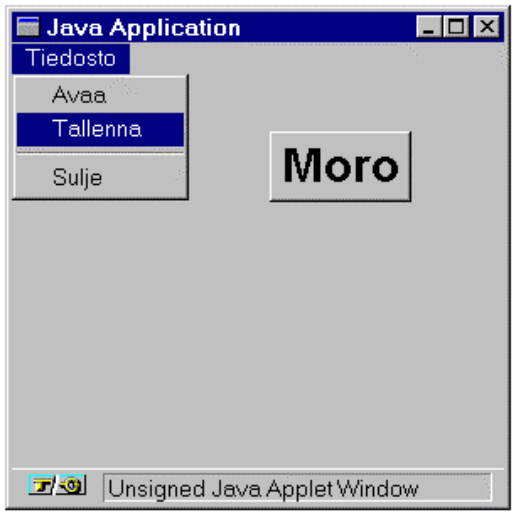
```
public void Ikkuna(String nimi) // muodostin
{
  setTitle(nimi);
  setVisible(true);
}
```

Luokkaan voidaan ylikirjoittaa helposti kaikki tarvittavat takaisinkutsuttavat funktiot, esimerkiksi tapahtumafunktiot. Eräs tärkeä tapahtuma on ikkunan sulkeminen, jonka toimintaan saamiseksi on ylikirjoitettava **`windowClosing`**-funktio.

Ikkuna voidaan hävittää näkyvistä funktiolla `setVisible`, kun parametriksi annetaan false. Jos ikkunaan ei enää tarvita, voidaan se poistaa käytöstä **`dispose`**-funktiolla. Yleensä ikkuna poistetaan näkyvistä sillloin, kun käyttäjä haluaa sulkea ikkunan sulje-painikkeesta.

```
public void windowClosing(WindowEvent e)
{
  setVisible(false);
  dispose();
}
```

Ikkunassa voidaan suorittaa kaikkia samoja graafisia toimenpiteitä kuin paneeleissakin. Ikkunoihin voidaan liittää graafisia elementtejä, piirtää grafiikkaa ja tulostaa tekstiä. Lisäksi ikkunaan voidaan liittää valikko, josta käyttäjä voi valita haluamansa toimenpiteet, esimerkiksi Tallenna-valikkokomento. Katso viereinen kuva.



Ikkunaan pätevät samat periaatteet kuin paneeleihin - containeriin liitetyt komponentit tulevat heti näkyviin.
Java-ohjelma **Ikkuna.java** esittelee ikkunan luomista erillisen luokan avulla. Esimerkissä on on kaksi luokkaa, Ikkuna ja Ikkunaluokka. Ikkuna on appletti ja se kutsuu IkkunaLuokka nimistä luokkaa, joka avaa ikkunan.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class ikkuna extends Applet
{
 public void init()
 {
  new IkkunaLuokka("Ikkuna");
 }
}

class IkkunaLuokka extends Frame implements
WindowListener
{
 public IkkunaLuokka(String nimi)
 {
  addWindowListener(this);
  setTitle("Oma ikkuna");
  setVisible(true);
```

```
  setSize(400, 300);
 }

 public void windowClosing(WindowEvent e)
 {
  setVisible(false);
  dispose();
 }

 public  void windowOpened(WindowEvent e)
 {
 }

 public void windowClosed(WindowEvent e)
 {
 }

 public void windowIconified(WindowEvent e)
 {
 }

 public void windowDeiconified(WindowEvent e)
 {
 }
 public void windowActivated(WindowEvent e)
 {
 }
 public void windowDeactivated(WindowEvent e)
 {
 }
}
```

Ikkunalle saadaan paljon sellaisia ominaisuuksia, joita esimerkiksi paneelilla ei voi olla. Ikkunalla voi olla esimerkiksi valikko, josta valitaan komentoja. Hiiren kohdistimen ei tarvitse olla sama nuolikohdistin, vaan se voidaan vaihtaa **`setCursor`**-funktiolla. SetCursor**-**funktio sijaitsee 1.1-versiossa Component-luokassa. Ikkunalle voidaan antaa myös kuvake funktiolla **`setIconImage`**.

dispose()	Vapauttaa ikkunan kuluttamat resurssit
getCursor()	Palauttaa hiiren kohdistimen
getIconImage()	Palauttaa kuvakkeen
getMenuBar()	Palauttaa nykyisen valikon
getTitle()	Palauttaa ikkunan nimen

isResizable()	Tosi, jos ikkunan kokoa voi muuttaa
setCursor(Cursor)	Asettaa hiiren kohdistimen
setIconImage(Image)	Asettaa ikkunalle kuvakkeen (gif-kuva)
setMenuBar(MenuBar)	Asettaa valikkopalkin
setResizable(boolean)	Sallii/estää ikkunan koon muutoksen
setTitle(String)	Asettaa ikkunan nimen
remove(MenuComponent)	Poistaa valikkopalkin

Ikkunan kuvake (*icon*) on GIF-kuva, ja se voi olla melkein mikä tahansa Internetistä ladattu kuva, tosin käyttöjärjestelmä rajoittaa kuvakkeen kokoa (Windowsissa 16*16 pikseliä). Javan 1.0-versiossa kuvake ei toimi, mutta sen sijaan 1.1-versiossa se toimii.

```
setIconImage(image);
```

Hiiren kohdistin on käyttöjärjestelmään sisältyvä valmis kuva, jolle on annettu nimi. Kohdistimen nimi on yksi seuraavista.

DEFAULT_CURSOR	Nuolikohdistin
CROSSHAIR_CURSOR	Ristikohdistin
HAND_CURSOR	Käsikohdistin
MOVE_CURSOR	Siirtokohdistin
TEXT_CURSOR	Tekstikohdistin eli I-kohdistin
WAIT_CURSOR	Odotuskohdistin eli tiimalasi
E_RESIZE_CURSOR	Koon muutos: itä
NE_RESIZE_CURSOR	Koon muutos: koillinen
NW_RESIZE_CURSOR	Koon muutos: luode
N_RESIZE_CURSOR	Koon muutos: pohjoinen
SE_RESIZE_CURSOR	Koon muutos: kaakko
SW_RESIZE_CURSOR	Koon muutos: lounas
S_RESIZE_CURSOR	Koon muutos: etelä
W_RESIZE_CURSOR	Koon muutos: länsi

HUOM! Kohdistimet löytyvät Frame-luokasta, mutta niitä ei kannata käyttää, koska Frame-luokan SetCursor-funktio on poistettu käytöstä. Kohdistimien nimet löytyvät Cursor-luokasta.

Kun ikkuna tehdään omassa luokassa, voidaan kohdistin asettaa yksinkertaisesti näin:

```
setCursor(Cursor.HAND_CURSOR);
```

Window-luokassa on lisäksi kaksi funktiota, joita voi käyttää ikkunan esiintuomisessa sekä taakseviemisessä.

toBack()	Vie ikkuna alimmaiseksi
toFront()	Tuo ikkuna päällimmäiseksi

Java-ohjelma **ikkuna2.java** esittelee ikkunan luomista erillisen luokan avulla. Lisäksi ohjelmassa käsitellään yllä olevien Frame- ja Window-luokkien funktioita.

```
import java.awt.*;
import java.applet.*;
import java.net.*;
import java.awt.event.*;

public class Ikkuna2 extends Applet
{
 Image im;
 public void init()
 {
  im = getImage(getDocumentBase(),"ikoni.gif");
  new IkkunaLuokka2("Ikkuna", im);
 }
}

class IkkunaLuokka2 extends Frame implements
WindowListener
{
 public IkkunaLuokka2(String nimi, Image im)
 {
  addWindowListener(this);
  setIconImage(im);
  setTitle("Ikkuna");
  setCursor(new Cursor(CROSSHAIR_CURSOR));
  setResizable(false);
  setVisible(true);
  setSize(300, 200);
 }
 public void windowClosing(WindowEvent e)
```

```
 {
  setVisible(false);
  dispose();
 }

 public  void windowOpened(WindowEvent e)
 {
 }

 public void windowClosed(WindowEvent e)
 {
 }

 public void windowIconified(WindowEvent e)
 {
 }

 public void windowDeiconified(WindowEvent e)
 {
 }

 public void windowActivated(WindowEvent e)
 {
 }

 public void windowDeactivated(WindowEvent e)
 {
 }

}
```

8.2 Ikkunoiden toiminnot

Ikkunoihin liittyvät tapahtumia ohjataan tapahtumaluokalla **WindowEvent**. Luokka sisältää vain yhden tarpeellisen funktion **`getWindow`**, jolla luetaan tapahtuman aiheuttaneen ikkuna-luokan esiintymä.

getWindow()	Palauttaa tapahtuman aiheuttaneen ikkuna-luokan esiintymän

Erilaisia tapahtumia on olemassa seitsemän. Nämä kattavat melko hyvin kaikki mahdolliset käyttäjän tekemät toimenpiteet. Tapahtumien tyypit ovat seuraavat.

WINDOW_ACTIVATED	Ikkuna aktivoitu (järjestelmä suorittaa)
WINDOW_CLOSED	Ikkuna suljettu
WINDOW_CLOSING	Ikkuna suljetaan (käyttäjä haluaa sulkea ikkunan)
WINDOW_DEACTIVATED	Ikkuna deaktivoitu (järjestelmä suorittaa)
WINDOW_DEICONIFIED	Ikkuna normaalisoitu (järjestelmä suorittaa)
WINDOW_ICONIFIED	Ikkunan pienennetty kuvakkeeksi (järjestelmä suorittaa)
WINDOW_OPENED	Ikkuna avautunut (järjestelmä suorittaa)

Osa toiminnoista ovat sellaisia, että käyttöjärjestelmä suorittaa tarvittavat toimenpiteet. Esimerkiksi, kun käyttäjä painaa ikkunassa olevaa pienennä-painiketta, käyttöjärjestelmä pienentää ikkunan automaattisesti. Toimenpiteestä annetaan vain ilmoitus ohjelmalle (tässä tapauksessa se olisi WINDOW_ICONIFIED). Kun käyttäjä haluaa sulkea ikkunan, lähetään tapahtuma WINDOW_DESTROY, mutta ikkunaa ei suljeta. Se on ohjelman tehtävä!

Tapahtumien tarkkailuun on olemassa **WindowListener**-rajapinta ja **WindowAdapter**-luokka. Yllä esitetyille tapahtumille omat takaisinkutsuttavat funktiot sekä WindowListener-rajapinnassa että sovitinluokassa. Takaisinkutsuttavat funktiot ovat muotoa:

```
public void windowOpened(WindowEvent e)
public void windowClosing(WindowEvent e)
public void windowClosed(WindowEvent e)
public void windowIconified(WindowEvent e)
public void windowDeiconified(WindowEvent e)
public void windowActivated(WindowEvent e)
public void windowDeactivated(WindowEvent e)
```

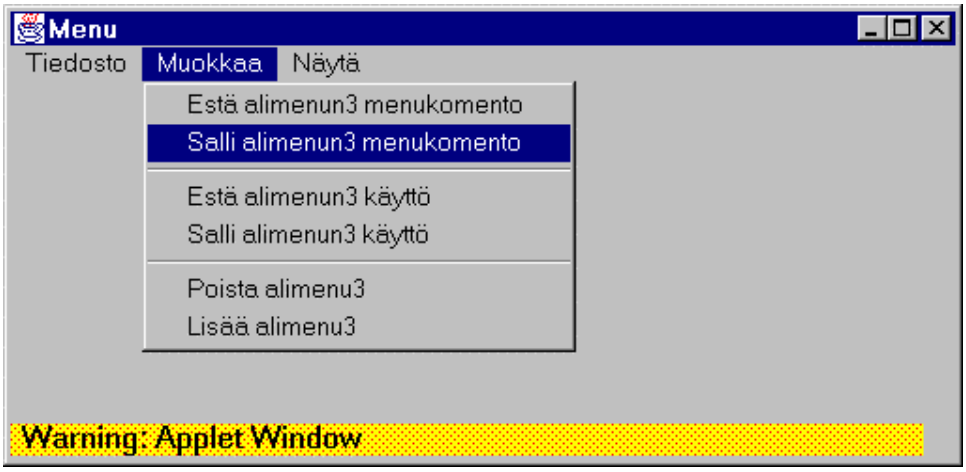
WindowEvent-olio sisältää vain sen tiedon mikä ikkuna tapahtuman aiheutti. Tapahtumatyypin määrää takaisinkutsuttavafunktio. Esimerkiksi, jos halutaan tietää mikä ikkuna pienennettiin kuvakkeeksi menetellään näin (tiedosssa pitää vertailtavan ikkuna-luokan esiintymä):

```
public void windowIconified(WindowEvent e)
{
```

```
 if (e.getWindow().equals(ikkuna1))
  // ikkuna1 pienennettiin
 else
  // oli joku muu
}
```

8.3 Valikot

Valikko eli valikko on varmasti tuttu jokaiselle. Valikkoihin liittyvät termit sen sijaan saattavat olla hieman outoja, joten käydään niitä hieman läpi seuraavan kuvan avulla. Kuvassa on ikkuna, jossa on otsikkopalkki (= kertoo ikkunan nimen) ja valikkopalkki. Valikkopalkissa on otsikoita (mm. Tiedosto, Muokkaa ja Näytä). Näitä otsikoita kutsutaan alivalikoiksi. Alivalikot puolestaan koostuvat valikkokomennoista. Kuvassa on valittuna alivalikko Muokkaa, jossa on 6 valikkokomentoa.



Javassa valikko tehdään aina ikkunaan eli Frame-olioon. Valikon tekemiseen tarvitaan useita luokkia, joista kukin tekee yhden pienen osan suuremmasta kokonaisuudesta. Voidaan eritellä osat valikkokomento, alivalikko ja valikkopalkki. Niitä vastaavat luokat on esitetty alla.

Valikkokomento	MenuItem-luokka
Alivalikko	Valikko-luokka
Valikkopalkki	MenuBar-luokka

Valikkoa tehtäessä menetellään yläpuolella esitetyssä järjestyksessä: ensiksi luodaan valikkokomennot,

```
komento1 = new MenuItem(”Avaa”));
komento2 = new MenuItem(”Sulje”));
```

sitten alivalikko, johon valikkokomennot liitetään

```
menu = new Menu(”Tiedosto”));
menu.add(komento1);
menu.add(komento2);
```

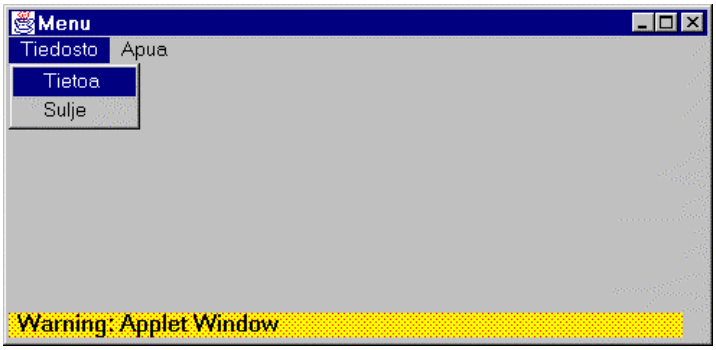
ja lopuksi valikkopalkki, johon liitetään alivalikko.

```
palkki = new MenuBar());
palkki.add(menu)
```

Ennen kuin valikko saadaan näkyviin on suoritettava funktiokutsu **`setMenuBar`**. Tämän jälkeen valikkopalkki kaikkine alivalikkoineen piirtyy ikkunaan.

```
ikkuna.setMenuBar(palkki);
```

Java-ohjelma **Menu1.java** esittelee valikon luontia omaan ikkunaan. Ohjelmassa käytetään kahta luokkaa Menu1 ja MenuIkkuna1, joista jälkimmäinen edustaa todellista ikkunaa, jonka ensimmäinen luokka avaa. MenuIkkuna-luokan muodostimessa liitetään ikkunaan valikkopalkki, jossa on alivalikot Tiedosto ja Apua. Tiedosto-alivalikon alle luodaan valikkokomennot Tietoa ja lopeta (katso kuva). Apua-alivalikon alle luodaan valikkokomento Apua. Valikkokomentoihin ei ole kytketty toimintoja.



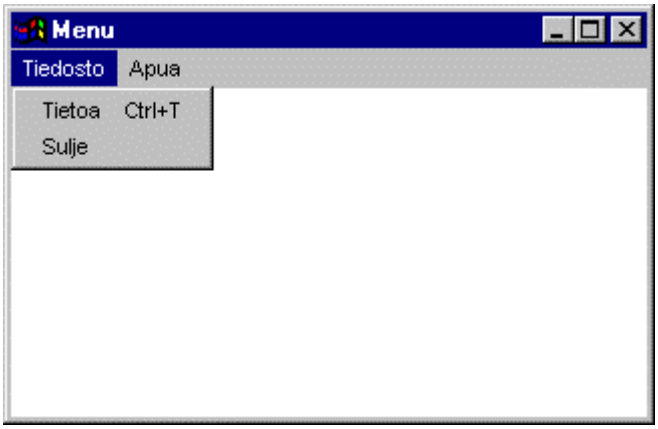

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Menu1 extends Applet
{
```

```
 public void init()
 {
  new MenuIkkuna1("Menu");
 }
}
class MenuIkkuna1 extends Frame implements WindowListener
{
 MenuBar menubar;
 Menu menu1, menu2;
 MenuItem apua;

 public MenuIkkuna1(String nimi)
 {
  addWindowListener(this);

  // Tee menubar-olio ja lisää siihen alimenut
  menubar = new MenuBar();
  menubar.add(menu1 = new Menu("Tiedosto"));
  menubar.add(menu2 = new Menu("Apua"));

  // Lisää alimenuihin menukomennot
  menu1.add (new MenuItem("Tietoa"));
  menu1.add (new MenuItem("Sulje"));

  menu2.add (apua = new MenuItem("Apua"));

  // aseta menupalkki näkyviin ja näytä ikkuna
  setMenuBar(menubar);
  setTitle(nimi);
  setVisible(true);
  setSize(200, 100);
 }

 public void windowClosing(WindowEvent e)
 {
  setVisible(false);
  dispose();
 }

 public  void windowOpened(WindowEvent e)
 {
 }

 public void windowClosed(WindowEvent e)
 {
 }
```

```
  public void windowIconified(WindowEvent e)
  {
  }

  public void windowDeiconified(WindowEvent e)
  {
  }

  public void windowActivated(WindowEvent e)
  {
  }

  public void windowDeactivated(WindowEvent e)
  {
  }

}
```

8.4 Valikot ja näppäimistö

Valikkokomentoja voi valita nyt myös näppäimistöltä. Versiossa 1.0 valikkokomentojen valinta suoritettiin aina hiirellä. Valikkokomennoissa esiintyy valikkonvalintanäppäin (*accelerator*). Esimerkiksi alla olevassa kuvassa Tiedosto-alivalikon valikkokomennon Tietoa valikonvalintanäppäin on CTRL+T.



HUOM! Valikonvalintanäppäin valitaan ohjelmassa käyttöjärjestelmäkohtaisen näppäimen avulla. Windowsissa ja Motifissa se on CTRL-näppäin ja Macissa Command-näppäin. Valikonvalintanäppäimen voi lukea Toolkit-luokan funktiolla `getMenuShortcutKeyMask`.

Valikonvalintanäppäin muodostetaan oman MenuShortcut-luokan avulla ja se liitetään myöhemmin haluttuun valikkokomentoon. MenuShortcut-luokalla on kaksi muodostinta. Kummatkin muodostimet tarvitsevat parametrikseen valikonvalintanäppäimen. Valikonvalintanäppäin on virtuaalinäppäinkoodi (esim. VK_X), joka löytyy KeyEvent-luokasta.

MenuShortcut(int)	Muodostaa MenuShortcut-luokan, jolla on tietty valikonvalintanäppäin.
MenuShortcut(int, boolean)	Muodostaa MenuShortcut-luokan, jolla on tietty valikonvalintanäppäin ja joka voidaan valita SHIFT-näppäimen avulla.

Luokan tärkeimmät funktiot ovat:

getKey()	Palauttaa valikonvalintanäppäimen
usesShiftModifier()	Tosi, jos valikonvalintanäppäin valitaan SHIFT-näppäimen kanssa.

Valikonvalintanäppäimen liittäminen valikkokomentoon tapahtuu MenuItem-luokan funktiolla **`setShortcut`** tai MenuItem-luokan muodostamisen yhteydessä. Jos valikonvalintanäppäin liitetään myöhemmin, käytetään setShortcut-funktiota.

```
komento1 = new MenuItem(”Avaa”));
.
.
s = new (MenuShortcut(KeyEvent.VK_A));
komento1.setShortcut(s);
```

Toinen tapa on rekisteröidä valikonvalintanäppäin heti muodostimessa.

```
s = new MenuShortcut(65));
komento1 = new MenuItem(”Avaa”), s);
```

HUOM! Valikonvalintanäppäimen käyttö edellyttää, että valikkokomennolle on asetettu **`setActionCommand`**-funktiolla tapahtumanimi.

```
Komento1.SetActionCommand(”Avaa”);
```

Esimerkki valikonvalintanäppäimen käytöstä on **menu2.java**. Esimerkki esittelee valikkokomentojen suorittamista. Koska valikonvalintanäppäin antaa saman toiminnon kuin hiirellä valittaessa, erillistä esimerkkiä ei ole.

8.5 Popup-valikko

Popup-valikko on tyypillinen nykyään graafisissa käyttöjärjestelmissä. Popup-valikko ponnahtaa näkyviin esimerkiksi Windowsissa hiiren oikean napin painalluksesta. Muissa käyttöjärjestelmissä hieman erilaisesti. Java kuitenkin tarjoaa laiteriippumattoman tavan popup-valikkojen käyttöön. Kuvassa Windowsin tuottama popup-valikko.

Popup-valikko eroaa tavallisesta valikosta siten, että se ei tarvitse lainkaan valikkopalkkia eikä alivalikkoja. Popup-valikko itsessään on yksi alivalikko, jossa on useita ei valikkokomentoja. Popup-valikkoa tehdessä siis pitää rekisteröidä yksi PopMenu-luokan esiintymä ja siihen vähintään yksi valikkokomento (=MenuItem-luokan esiintymä).

```
PopupMenu popup;
MenuItem avaa;

popup = new PopupMenu();
avaa = new MenuItem("Avaa");
popup.add(avaa);
```

Javassa popup-valikot on sidottu komponentteihin. Jokaisella popup-valikolla pitää olla äiti-komponentti. Se voi olla painonappi tai vaikkapa ikkuna. Popup-valikko liitetään add-funktiolla komponenttiin ennenkuin sitä voidaan käyttää. Remove-funktiolla popup-valikko puolestaan poistetaan komponentista, kun sitä ei enää tarvita.

```
add(popup);
```

HUOM! Vain yksi komponentti voi omistaa yhden popup-valikon kerrallaan. Mikäli toinen komponentti haluaa käyttää popup-valikkoa itsellään, pitää popup-valikko ensiksi poistaa remove-funktiolla toisesta komponentista.

Valikko saadaan näkyviin, kun kutsutaan **show**-funktiota. Funktio tarvitsee parametrikseen komponentin ja koordinaatit. Komponentti on se, johon add-funktiolla popup-valikko on liitetty tai tästä komponentista periytynyt luokka. Täten myös ikkuna voi omistaa popup-valikon.

```
show(this, 10, 10);
```

Valikko tulee näkyviin annetuihin koordinaatteihin suhteessa komponenttiin. Koordinaatit saadaan helpoimmin luettua suoraan hiiren tapahtumaoliosta MouseEvent. Tällöin popup-valikko aukeaa hiiren osoittimen viereen.

MouseEvent-luokan **isPopupTrigger**-funktiolla tutkitaan oliko kyseessä popup-valikon esiintuontikäsky. Funktio palauttaa arvon tosi, jos käyttäjä haluaa popup-valikko esiin. Ohjelman tehtävänä on tämän jälkeen kutsua edellä esitettyä show-funktiota. Funktiolla **e.getComponent** saadaan luettua komponentti, joka aiheutti tapahtuman.

```
if (isPopupTrigger())
 show(e.getComponent(), e.getX(), e.getY());
```

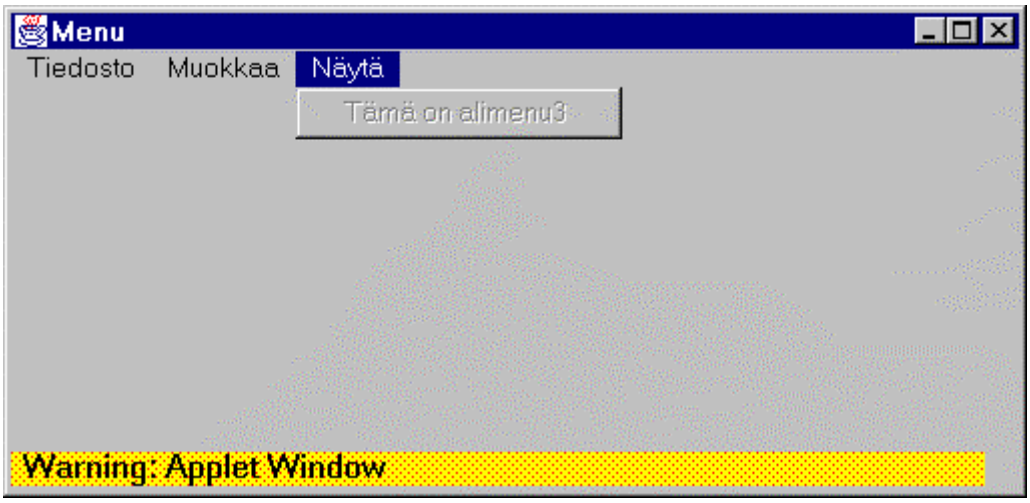
Seuraava esimerkki esittelee popup-valikon käyttöönottamista. Ohjelmassa käsitellään hiiren tapahtumat **processMouseEvent**-funktioilla. Tämä siksi, että MouseListener-rajapinnan kautta voidaan reagoida vain vasemman napin painalluksiin.

```
public class PopupMenuTest extends Applet
{
 public void init()
 {
  new PopupMenuFrame();
 }
}

class PopupMenuFrame extends Frame
{
  PopupMenu popup;

  public PopupMenuFrame()
  {
   popup = new PopupMenu("Muokkaa");
   popup.add(new MenuItem("Leikkaa"));
   popup.add(new MenuItem("Kopioi"));
   popup.add(new MenuItem("Liitä"));
   popup.addSeparator();
```

```
    popup.add(new MenuItem("Fontti"));
    add(popup);

    enableEvents(AWTEvent.MOUSE_EVENT_MASK);
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);

    setTitle("PopupMenu");
    setSize(300, 200);
    setVisible(true);
  }

  public void processMouseEvent(MouseEvent e)
  {
    if (e.isPopupTrigger())
      {
       popup.show(e.getComponent(), e.getX(), e.getY());
      }
    super.processMouseEvent(e);
  }

  public void processWindowEvent(WindowEvent e)
  {
    if (e.getID() ==  e.WINDOW_CLOSING)
    {
     setVisible(false);
     dispose();
    }
    super.processWindowEvent(e);
  }
}
```

8.6 Valikon muokkaaminen

Kun valikko on luotu, sitä voidaan käsitellä luokkien (MenuItem, Menu ja MenuBar) funktioilla. Esimerkiksi valikkokomennon Y toiminta voidaan estää, kun valikkokomento X aktivoidaan. Alla olevassa kuvassa on estetty valikkokomento ”Näytä->Tämä on alimenu 3”. Lisäksi luokkien funktioilla voidaan poistaa alivalikkoja, lisätä uusia valikkokomentoja ja niin edelleen.



Seuraavaksi tutustutaan järjestyksessä (MenuItem, Menu, MenuBar) kunkin luokan funktioihin.

MENUITEM-luokka (eli valikkokomento)

deleteShortcut()	Tuhoaa valikonvalintanäppäimen
getLabel()	Lukee valikkokomennon nimen
getShortcut()	Lukee valikonvalintanäppäimen
isEnabled()	Tutkii, onko valikkokomento sallittu
setEnabled()	Sallii/estää valikkokomennon käytön
setLabel(String)	Asettaa valikkokomennon nimen
setShortcut(MenuShortcut)	Asettaa valikonvalintanäppäimen

Esimerkiksi valikkokomennon Apua estäminen tapahtuu näin:

```
apua.setEnabled(false);
```

Huomaa, että valikkokomento on edellisessä esimerkissä määritelty näin:

```
menu2.add (apua = new MenuItem("Apua"));
```

MENU-luokka (eli alivalikko)

add(MenuItem)	Lisää valikkokomennon alivalikkoon
add(String)	Lisää merkkijonon alivalikkoon
addSeparator()	Lisää erottimen (vaakasuora viiva valikkokomentojen välissä) nykyiseen kohtaan
getItem(int)	Lukee valikkokomennon järjestysnumeron perusteella
getItemCount()	Laskee valikkokomennot
insert(MenuItem, int)	Lisää valikkokomennon tiettyyn paikkaan
insert(String, int)	Lisää merkkijonon tiettyyn paikkaan
insertSeparator(int)	Lisää erottimen tiettyyn paikkaan
isTearOff()	Tosi, jos tear-off-valikko
remove(int)	Tuhoaa valikkokomennon järjestysnumeron perusteella
remove(MenuComponent)	Tuhoaa tietyn valikkokomennon
removeAll()	Tuhoaa kaikki valikkokomennot

Esimerkiksi erottimen lisääminen alivalikkoon Tiedosto onnistuu näin:

```
menu1.addSeparator();
```

MENUBAR-luokka (valikkopalkki)

add(Menu)	Lisää alivalikon valikkopalkkiin
deleteShortcut(MenuShortcut)	Tuhoaa valikonvalintanäppäimen
getHelpMenu()	Palauttaa Apua-valikon
getMenuCount()	Palauttaa alivalikkojen määrän
getMenu(int)	Palauttaa alivalikon järjestysnumeron perusteella
remove(int)	Poistaa alivalikon valikkopalkista järjestysnumeron perusteella
remove(MenuComponent)	Poistaa tietyn valikkokomennon
setHelpMenu(Menu)	Asettaa Apua-valikon valikkopalkkiin

shortcuts()	Etsii kaikki valikonvalintanäppäimet

Esimerkiksi olemassa olevien alivalikkojen laskeminen suoritetaan funktiolla **`getMenuCount`**.

```
int alimenuja = menubar.getMenuCount();
```

8.7 Valikkokomennon suorittaminen

Tähän mennessä on opittu, kuinka valikkoja luodaan ja kuinka niitä muokataan, mutta vielä ei ole kerrottu kuinka valittu valikkokomento tunnistetaan. Tunnistaminen perustuu tapahtumaluokkaan **ActionEvent**. Luokka on jo ollut esillä luvussa *6.10 Interaktiivisuutta elementeillä*.

Jokaiselle valikkokomennolle pitää asettaa erikseen kuuntelija. Se tapahtuu tutulla **`addActionListener`**-funktiolla. Vastaavasti yhteys kuuntelijaan puretaan **`removeActionListener`**-funktiolla.

```
Item1 = new MenuItem(”Avaa”);
item1.AddActionListener(this);
```

Oletusnimenä tapahtuman aiheuttajalle käytetään valikkokomennon nimeä (yllä nimi on Avaa). Se voidaan kuitenkin asettaa toiseksi funktiolla **`setActionCommand`**, jos valikkokomennon nimi ei ole sopiva. Tapahtumanimi pitää asettaa, jos halutaan käyttää valikonvalintanäppäintä. Funktiolla **`getActionCommand`** voidaan puolestaan lukea tapahtumanimi.

setActionCommand(String)	Asettaa tapahtumanimen
getActionCommand()	Lukee tapahtumanimen

ActionListener-rajapinnassa on funktio **`actionPerformed`**, jota kutsutaan silloin kun valikkokomento on valittu**.** Funktio palauttaa ActionEvent-luokan olion. Olion funktion **`getActionCommand`** avulla voidaan lukea mikä olio tapahtuman aiheutti.

```
public void actionPerformed(ActionEvent e)
{
 if (e.getActionCommand().equals(”Sulje”))
 {
  setVisible(false);
  dispose();
 }
}
```

Java-ohjelma **Menu2.java** esittelee valikkojen luontia, muokkaamista ja valikkokomentojen suorittamista. Ohjelmassa käytetään kahta luokkaa Menu2 ja MenuIkkuna2, joista jälkimmäinen edustaa ikkunaa, jonka ensimmäinen luokka avaa. Tapahtumafunktio actionPerformed vastaa valikkokomentojen suorittamisesta. Ohjelman alivalikossa Muokkaa on valikkokomentoja, jotka vaikuttavat Näytä-alivalikon toimintaan.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Menu2 extends Applet
{
 public void init()
 {
  new MenuIkkuna2("Menu");
 }
}

class MenuIkkuna2 extends Frame implements
ActionListener, WindowListener
{
 MenuBar menubar;
 Menu menu1, menu2, menu3;
 MenuItem menukomento;


 public MenuIkkuna2(String nimi)
 {
  MenuItem mi;
  addWindowListener(this);

  // Tee menubar-olio ja lisää siihen alimenut
  menubar = new MenuBar();
  menubar.add (menu1 = new Menu("Tiedosto"));
  menubar.add (menu2 = new Menu("Muokkaa"));
  menubar.add (menu3 = new Menu("Näytä"));

  // Lisää alimenuihin menukomennot
  menu1.add (new MenuItem("Avaa"));
  menu1.getItem(0).addActionListener(this);
  menu1.add (new MenuItem("Tallenna"));
  menu1.getItem(1).addActionListener(this);

  menu1.addSeparator();
```

```
 menu1.add (new MenuItem("Sulje"));
 menu1.getItem(3).addActionListener(this);


 menu2.add (new MenuItem("Estä alimenun3 menukomento",
                      new MenuShortcut(KeyEvent.VK_E)));
 menu2.getItem(0).addActionListener(this);
 menu2.getItem(0).setActionCommand("estakomento");
 menu2.add (new MenuItem("Salli alimenun3 menukomento",
                       new MenuShortcut(KeyEvent.VK_S)));
 menu2.getItem(1).addActionListener(this);
 menu2.getItem(1).setActionCommand("sallikomento");

 menu2.addSeparator();

 menu2.add (new MenuItem("Estä alimenun3 käyttö"));
 menu2.getItem(3).addActionListener(this);
 menu2.getItem(3).setActionCommand("estamenu");
 menu2.add (new MenuItem("Salli alimenun3 käyttö"));
 menu2.getItem(4).addActionListener(this);
 menu2.getItem(4).setActionCommand("sallimenu");

 menu2.addSeparator();

 menu2.add (new MenuItem("Poista alimenu3"));
 menu2.getItem(6).addActionListener(this);
 menu2.getItem(6).setActionCommand("poistamenu");
 menu2.add (new MenuItem("Lisää alimenu3"));
 menu2.getItem(7).addActionListener(this);
 menu2.getItem(7).setActionCommand("lisaamenu");


 menu3.add (menukomento = new MenuItem("Tämä on
                                       alimenu3"));
 menu3.getItem(0).addActionListener(this);
 menu3.getItem(0).setActionCommand("alimenu3");


 setMenuBar(menubar);    // aseta menupalkki ikkunaan
 setTitle(nimi);
 setVisible(true);
 setSize(300, 200);

}

public void actionPerformed(ActionEvent e)
{
```

```
   if (e.getActionCommand().equals("Sulje"))
   {
    setVisible(false);
    dispose();
   }

   if (e.getActionCommand().equals("estakomento"))
    menukomento.setEnabled(false);

   if (e.getActionCommand().equals("sallikomento"))
    menukomento.setEnabled(true);

   if (e.getActionCommand().equals("estamenu"))
    menu3.setEnabled(false);

   if (e.getActionCommand().equals("sallimenu"))
    menu3.setEnabled(true);

   if (e.getActionCommand().equals("poistamenu"))
    menubar.remove(menu3);

   if (e.getActionCommand().equals("lisaamenu"))
    if (menubar.getMenuCount() != 3)
    {
     menubar.add (menu3 = new Menu("Näytä"));
     menu3.add (new MenuItem ("Tämä on alimenu3"));
    }

 }


 public void windowClosing(WindowEvent e)
 {
  setVisible(false);
  dispose();
 }
 public void windowOpened(WindowEvent e) { }
 public void windowClosed(WindowEvent e) { }
 public void windowIconified(WindowEvent e) { }
 public void windowDeiconified(WindowEvent e) { }
 public void windowActivated(WindowEvent e) { }
 public void windowDeactivated(WindowEvent e) { }

}
```

8.8 Dialogit

Dialogilla tarkoitetaan selaimesta erillistä, itsenäistä ikkunaa, joka perustuu johonkin olemassa olevaan ikkunaan. Dialogilla ei ole välttämättä nimipalkkia (*titlebar*), mutta sillä on piirtoalue (*client area*), johon voi kohdistaa graafisia toimintoja kuten ikkunoihin ja paneeleihin. Graafisissa käyttöjärjestelmissä dialogeja käytetään käyttäjän ja ohjelman välisenä käyttöliittymänä, koska niiden avulla on helppo syöttää tietoa ohjelmaan. Dialogien käyttöliittymä muodostuu yleensä komponenteista, kuten painonapeista, listoista jne. Tyypillinen dialogi on tulostusdialogi.

```
+----java.awt.Component
        |
        +----java.awt.Container
              |
              +----java.awt.Window
                    |
                    +----java.awt.Dialog
```

Dialogi muodostetaan luokalla **Dialog**. Luokassa on kaksi muodostinta, molemmat tarvitsevat parametreikseen olemassa olevan ikkunan ja tiedon siitä onko dialogi tyyppiä **modal** tai **modeless**. Modal-tyyppinen dialogi estää muiden ikkunoiden käytön silloin kun dialogi on näkyvissä. Modeless-tyyppinen dialogi puolestaan sallii käyttäjän tehdä mitä tahansa toimenpiteitä dialogin ollessa näkyvissä. Appletviewer-ohjelma ei Java 1.0 -version kanssa osaa käsitellä dialogeja oikein. Virhe on korjattu versiossa 1.1.

Dialog(Frame, boolean)	Luo dialogin, joka on modal (tosi) tai modeless (epätosi)
Dialog(Frame, String, bool)	Luo dialogin, jolla on nimipalkki ja joka on tyypiltään modal (tosi) tai modeless (epätosi)

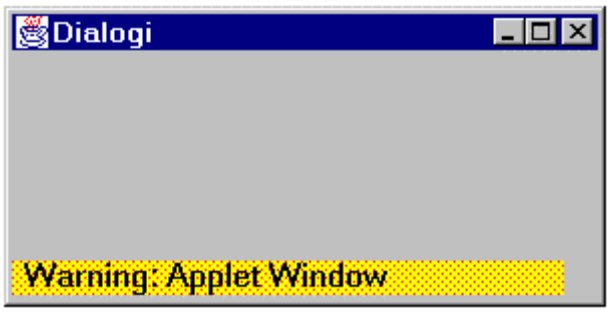
Dialogi voidaan luoda esimerkiksi näin:

```
Dialog dialogi = new Dialog(frame, ”Dialogi”, true);
```

Kun muodostinta on kutsuttu, järjestelmä luo dialogin, joka on piilotettu. Sen esiin tuomiseksi kutsutaan **`setVisible`**-funktiota parametrina **`true`**. Kutsumalla **`show`**-funktiota ikkuna tuodaan päälimmäiseksi, mikäli ikkuna on tuotu näkyviin setVisible-funktiolla. Dialog-luokalla ei ole hide-funktiota, joten dialogin piilottamiseksi on kutsuttava **`setVisible`**-funktiota parametrina **`false.`**

```
dialogi.setVisible(true);
```

Nyt dialogi avautuu kuvaruudulle. Dialogi on aivan kuin mikä tahansa muukin ikkuna, paitsi että sillä on alareunassa palkki, jossa on varoitusteksti. Teksti varoittaa käyttäjää siitä, että dialogin on luonut applet eikä selain. Varoitustekstiä ei voi mitenkään poistaa. Kuvassa appletviewerin tuottama dialogi.



Uuden dialogin avaaminen voidaan toteuttaa myös oman luokan avulla, kuten ikkunankin. Täten luokkaan voidaan ylikirjoittaa helposti kaikki tarvittavat takaisinkutsuttavat funktiot. Ohjelmasta tulee selkeämpi kun kaikki toiminnot kapseloidaan yhteen luokkaan.

```
class dialogi extends Dialog
```

Dialogi on ominaisuuksiltaan paljon vaatimattomampi kuin ikkuna (dialogilla on kuitenkin samat tapahtumat kuin ikkunanalla). Dialogin perusfunktioihin kuluu vain otsikon asettaminen ja ikkunan koon muutoksen sallinta/esto. Alla on esitetty Dialog-luokan funktioita.

getTitle()	Lukee dialogin otsikon
isModal()	Tosi, jos dialogi on modal-tyyppinen
isResizable()	Tosi, jos ikkunan kokoa voi muuttaa
setModal(boolean)	Asettaa dialogin modal-tyyppiseksi
setResizable(boolean)	Sallii/estää (tosi/epätosi) ikkunan koon muutoksen
setTitle(String)	Asettaa dialogin nimen

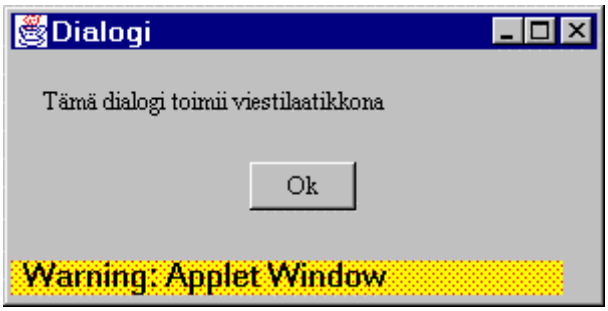
Window-luokassa on lisäksi kaksi funktiota, joita voi käyttää dialogin esiintuomisessa sekä taakseviemisessä.

toBack()	Vie dialogi alimmaiseksi

toFront()	Tuo dialogi päällimmäiseksi

Java-ohjelma **dialogi.java** esittelee dialogin ohjelmoimista käytännössä. Dialogi.java koostuu kahdesta luokasta: Dialogi-luokasta, joka on periytynyt Applet-luokasta sekä **MessageBox**-luokasta, joka on periytynyt Dialog-luokasta. Luokka MessageBox avaa Windows-tyylisen viestilaatikon.

Viestilaatikkoa kutsutaan siten, että muodostimessa annetaan ikkuna, jonka päälle viestilaatikko avautuu sekä otsikko ja ilmoitusteksti. Koska muodostin ei voi palauttaa arvoa, kutsutaan avaa-funktiota muodostimen yhteydessä. Funktio palauttaa sen arvon, joka aiheutuu painonapin painamisesta (Ok = 1 ja Peruuta = 2).




```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Dialogi extends Applet
{
 public void init()
 {
  Frame f = new Frame("Ikkuna");
  int tulos = new MessageBox(f,"Dialogi","Tämä dialogi
    toimii viestilaatikkona", 3).nayta();
 }
}

class MessageBox extends Dialog implements ActionListener
{
 public static final int OK = 1;
 public static final int PERUUTA = 2;
 int nappi;
 Panel p;
 Button ok, peruuta;
 MessageBox(Frame f, String otsikko, String teksti, int
    nappi)
 {
```

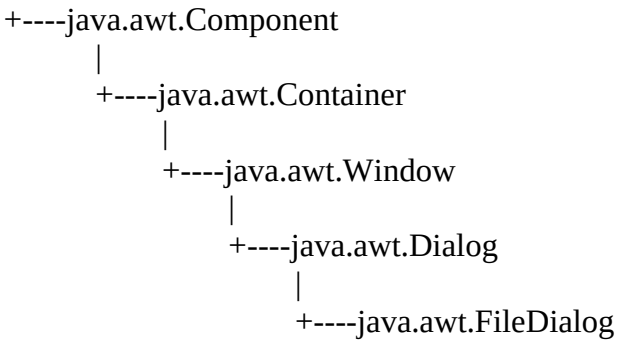
```
  super(f, otsikko, true);
  setLayout(new GridLayout(2,1));
  add (new Label(teksti));
  add (p = new Panel());
  p.setLayout(new FlowLayout());
  if ((nappi & 1) != 0)
    {
     p.add (ok = new Button("    Ok    "));
     ok.addActionListener(this);
     ok.setActionCommand("OK");
    }
  if ((nappi & 2) != 0)
    {
     p.add (peruuta = new Button(" Peruuta "));
         peruuta.addActionListener(this);
     peruuta.setActionCommand("PERUUTA");
    }
  setSize(300,150);
  setResizable(true);
 }

 public int avaa()
 {
  setVisible(true);
  return nappi;
 }

 public void actionPerformed(ActionEvent e)
 {
   if (e.getActionCommand().equals("OK"))
   {
    nappi = 1;
    setVisible(false);
    dispose();
   }
   if (e.getActionCommand().equals("PERUUTA"))
   {
    nappi = 2;
    setVisible(false);
    dispose();
   }
 }
}
```

8.9 Tiedostodialogit

Tiedostodialogit mahdollistavat valmiiden Avaa- ja Tallenna-dialogien käyttämisen. Tiedostodialogit perustuvat johonkin olemassa olevaan ikkunaan eli Frame-olioon. Tiedostodialogeilla on nimipalkki (*titlebar*), johon kirjoitetaan otsikko (Avaa tai Tallenna). Piirtoalue (*client area*) on varattu levyhakemiston kuvaamiseen.

Tiedostodialogit eivät ole tarkoitettu toimivaksi selain-ympäristösssä. Tosin selain saattaa avata tiedostodialogin, mutta tiedostojen avaamiseen ja tallentamiseen ne eivät käy. Dialogien ainoa käyttökohde onkin Java-sovellukset, jotka toimivat käyttöjärjestelmätasolla.

Tiedostodialogi on tyypiltään modal, joten muiden ikkunoiden osoittaminen ei onnistu ennenkuin dialogi suljetaan.



Tiedostodialogi muodostetaan luokalla **FileDialog**. Luokassa on kolme muodostinta, joista kaikki tarvitsevat parametreikseen olemassa olevan ikkunan. Kaksi jälkimmäistä tarvitsevat myös tiedostodialogin otsikon. Kolmannella muodostimella määrätään lisäksi, onko tiedostodialogi tarkoitettu lukuun (LOAD) tai tallennukseen (SAVE).

FileDialog(Frame)	Luo nimettömän tiedostodialogin
FileDialog(Frame, String)	Luo tiedostodialogin, jolla on otsikko
FileDialog(Frame, String, int)	Luo tiedostodialogin, jolla on otsikko sekä moodi (luku/tallennus)

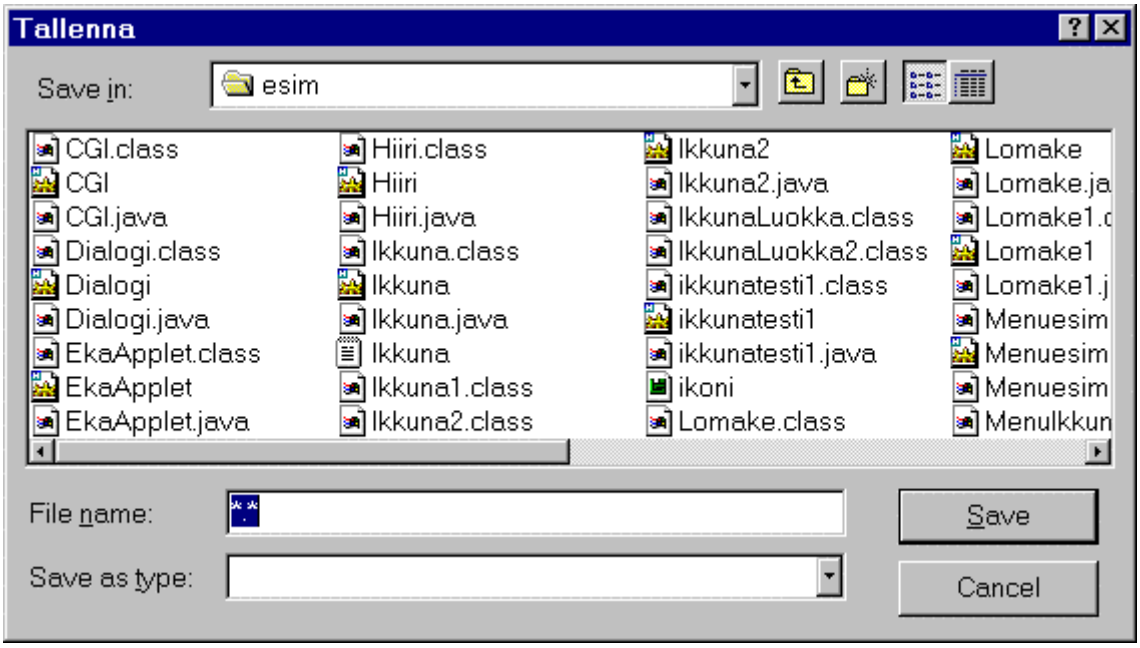
Esimerkiksi tallenna-dialogi voidaan luoda näin:

```
FileDialog tallenna = new FileDialog(frame,
                        ”Tallenna”, FileDialog.SAVE);
```

Kun muodostinta on kutsuttu, järjestelmä luo tiedostodialogin, joka on piilotettu. Sen esiin tuomiseksi kutsutaan **`setVisible`**-funktiota.
Kutsumalla **`show`**-funktiota tiedostodialogi tuodaan päälimmäiseksi, mikäli se on tuotu näkyviin setVisible-funktiolla.

```
tallenna.setVisible();
```

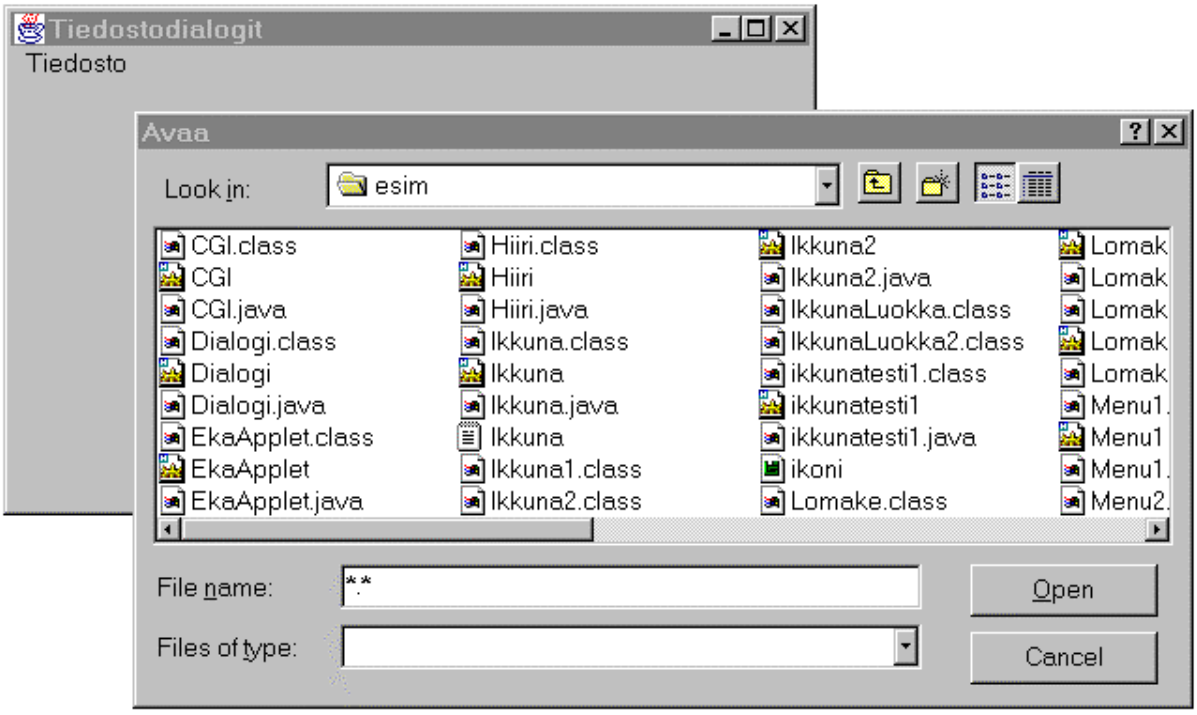
Nyt tiedostodialogi avautuu kuvaruudulle. Tiedostodialogi näyttää samanlaiselta kuin minkä tahansa muun ohjelman tuottama tiedostodialogi. Tiedostodialogilla ei ole varoitustekstiä, toisin kuin ikkunoilla ja dialogeilla.



Uuden tiedostodialogin avaaminen voidaan toteuttaa myös oman luokan avulla, kuten ikkunankin. Täten luokkaan voidaan ylikirjoittaa helposti kaikki tarvittavat takaisinkutsuttavat funktiot. Toisaalta tiedostodialogi voidaan toteuttaa myös yksinkertaisesti näin (katso myös lähdekoodia Menuesim.java)

```
new FileDialog(this, "Tallenna",
                    FileDialog.SAVE).show();
```

Java-ohjelma **TiedostoDialogi.java** esittelee tiedostodialogien käyttöä. Dialogit valitaan Tiedosto-alivalikon valikkokomennoilla Avaa ja Tallenna. Ohjelmassa käytetään kahta luokkaa TiedostoDialogi ja TiedostoIkkuna, joista jälkimmäinen edustaa ikkunaa, jonka ensimmäinen luokka avaa. Tapahtumafunktio action vastaa valikkokomentojen suorittamisesta. Ohjelma on suunniteltu siten, että se toimii selaimessa ja Javaa tukevissa käyttöjärjestelmissä. Kuvassa ohjelma on ajettu Java-tulkilla.




```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class TiedostoDialogi extends Applet
{
 public void init()
 {
  new TiedostoIkkuna("Tiedostodialogit");
 }

 public static void main(String args[])
 {
  TiedostoDialogi td = new TiedostoDialogi();
  td.init();
 }
}

class TiedostoIkkuna extends Frame implements
    ActionListener
{
 MenuBar menubar;
 Menu menu1, menu2, menu3;
 MenuItem menukomento;
```

```
 public TiedostoIkkuna(String nimi)
 {
  // Tee menubar-olio ja lisää siihen alimenut
  menubar = new MenuBar();
  menubar.add (menu1 = new Menu("Tiedosto"));

  // Lisää alimenuihin menukomennot
  menu1.add (new MenuItem("Avaa"));
  menu1.getItem(0).addActionListener(this);
  menu1.add (new MenuItem("Tallenna"));
  menu1.getItem(1).addActionListener(this);
  menu1.addSeparator();
  menu1.add (new MenuItem("Sulje"));
  menu1.getItem(3).addActionListener(this);

  setMenuBar(menubar);    // aseta menupalkki ikkunaan
  setTitle(nimi);
  show();
  setSize(300,150);
 }

 public void actionPerformed(ActionEvent e)
 {
  if (e.getActionCommand().equals("Sulje"))
    {
     setVisible(false);
     dispose();
    }
  if (e.getActionCommand().equals("Avaa"))
  {
    new FileDialog(this, "Avaa", FileDialog.LOAD).show();
  }

  if (e.getActionCommand().equals("Tallenna"))
  {
   new FileDialog(this,
    "Tallenna",FileDialog.SAVE).show();
  }
 }

}
```

8.10 Tulostaminen

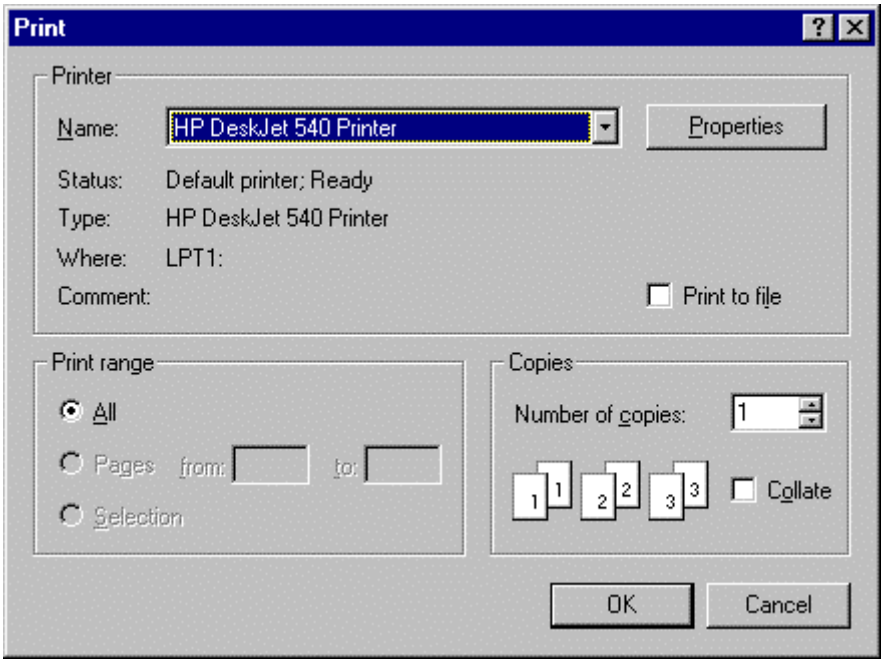
Javan 1.1-versio tarjoaa myös tulostusmahdollisuuden. Tulostus käyttää hyväkseen käyttöjärjestelmän omaa tulostusjärjestelmää ja tulostusdialogia, joten tulostaminen on nopeata ja yksinkertaista. Versio 1.1 ei kuitenkaan salli omien tulostusdialogien kirjoittamista.

Tulostaminen kohdistuu ikkunan piirtopintaan - kaikki mitä ikkunassa näkyy, voidaan tulostaa kirjoittimelle. Turvallisuussyistä applet ei voi tulostaa mitään. Siksi tulostaminen pitää hoitaa aina sovelluksesta!

Tulostustaminen Javassa on erittäin helppoa. Se tapahtuu kutsumalla java.awt.Toolkit-luokan **`getPrintJob`**-funktiota, joka palauttaa PrintJob-olion. Olio sisältää kaiken tiedon, jota tarvitaan tulostukseen.

```
PrintJob pjob = getToolkit().getPrintJob(this,
                "Printing Test", null);
```

Funktio getPrintJob pitää huolta, että käyttöjärjestelmäkohtainen tulostusdialogi avautuu näytölle. Funktion ensimmäiseksi parametriksi annettaan ikkuna, jonka päälle tulostusdialogi avataan. Kun käyttäjä hyväksyy dialogin, on PrintJob-olio asetettu vastaamaan käyttäjän toiveita. Ominaisuudet-parametri sallii syöttää mitä tahansa järjestelmäriippuvaisia ominaisuuksia tulostusta varten, esimerkiksi kirjoittimen nimen.



Jos käyttäjä vaihtaa oletusasetuksia tulostusdialogissa, muutokset tallennetaan Properties-olioon ja syötetään parametrina PrintJob-olioon.

PrintJob-olio tuottaa asiaankuuluvat tulostusominaisuudet, jotka käyttäjä valitsee (esimerkiksi sivun koko).

Kirjoittimella ja näytöllä on omat Graphics-oliot. Graphics-oliot edustavat näytön piirtopintaa. PrintJob-oliolla on funktio **`getGraphics`**, jolla saadaan luettua kirjoittimen Graphics-olio. Kirjoittimen Graphics-olio toteuttaa lisäksi java.awt.PrintGraphics-rajapinnan.

```
Graphics pg = pjob.getGraphics();
```

Kun Graphics-olio on saatu luettua, voidaan tulostaminen aloittaa. Component-luokassa kaksi tulostamiseen tarvittavaa funktiota:

print(Graphics) Tulostaa komponentin. Print-funktio kutsuu paint-funktiota, eli jos ohjelman joku komponentti piirtää itsensä paint-funktiossa, tulostaminen onnistuu automaattisesti.

printAll(Graphics) Tulostaa komponentin ja sen kaikki alikomponenetit

Jos halutaan tulostaa pelkästään yhden komponentin sisältö, esimerkiksi painonapin b, menetellään näin:

```
b.print(pg);
```

Tai mikäli halutaan tulostaa koko ikkunan sisältö (kaikki komponentit, jotka ikkunaan on liitetty). Kaikki komponentit kykenevät piirtämään itsensä, kun niiden printAll-funktiota kutsutaan.

```
printAll(pg);
```

Tulostus alkaa, kun kutsutaan **`dispose`**-funktiota. Käytännössä tällä funktiolla tuhotaan Graphics-olio.

```
pg.dispose();
```

Kun tulostus on päättynyt, kutsutaan end-funktiota. Funktio tekee tarvittavat toimenpiteet siivotakseen jälkensä (työntää paperin ulos jne).

```
pjob.end();
```

Koska kirjoittimen Graphics-olio on Graphics-luokan esiintymä, voidaan sitä käyttää myös paint-funktiossa. Käytännössä tämä tarkoittaa sitä, että paint-funktion aikana voidaan kirjoittimen piirtopinnalle piirtää Graphics-olion funktioilla grafiikkaa, tekstiä jne. Instanceof-lauseella voidaan tutkia paint-funktiossa kumman Graphics-olion se palauttaa (eli piirretäänkö näytölle vai tulostimelle).

```
public void paint(Graphics g)
{
  if (g instanceof PrintGraphics)
  {
   // tulostetaan kirjoittimelle
  }
  else
  {
   // tulostetaan näytölle
  }
}
```

Sivutus on ohjelman vastuulla. Jokainen getGraphics-funktiokutsu palauttaa kirjoittimen Graphics-olion yhdelle sivulle. Kun ohjelma kutsuu dispose-funktiota, tämä sivu siirretään kirjoittimelle. Ohjelman pitää huolta sivujen sisällöstä, sivun loppumisesta, paperin koosta ja tulostus resoluutiosta. Näitä tulostusoptioita käsittelee seuraavat funktiot.

getPageDimension()	Palautta paperin mitat Dimesion-oliossa.
getPageResolution()	Palauttaa paperin resoluution

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Tulosta extends Frame implements
ActionListener
{
  public Tulosta()
  {
   super("Tulostus");
   Button b = new Button ("Aloita tulostus");
   add("North",b);
   b.addActionListener(this);
```

```
    setSize(300, 300);
   }

   public static void main(String args[])
   {
    tulosta ikkuna = new tulosta();
    ikkuna.setVisible(true);
   }

  public void actionPerformed(ActionEvent e)
  {
    PrintJob pjob = getToolkit().getPrintJob(this,
     "Tulostus", null);
    Graphics pg = pjob.getGraphics();
    printAll(pg);
    pg.dispose(); // paperi ulos
    pjob.end();
}

  public void paint(Graphics g)
  {
    g.fillOval(60,40,120,200);
  }
}
```

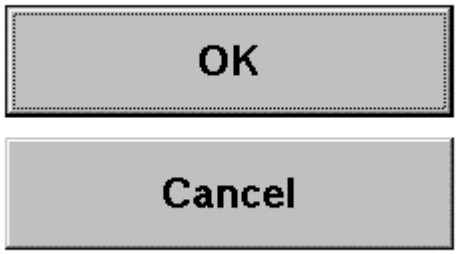
9. Graafiset elementit

Tässä luvussa käydään läpi graafisia elementtejä, joiden avulla käyttäjä vaikuttaa ohjelman kulkuun. Elementtejä ovat muun muuassa painonapit, valintalaatikot ja valintalistat.

HTML-kieli tarjoaa helpon tavan elementtien muodostamiseen, mutta niin tarjoaa myös Java (HTML-kieltä ei voi hyödyntää osana appletteja). Javassa on myös se etu, että elementtien käyttöä ei ole rajoitettu mitenkään, esimerkiksi interaktiivisuuden tuottaminen ei vaadi CGI-ohjelmien käyttöä.

9.1 Painonappi

Painonappi (*button*) on tyypillinen kaikissa graafisissa sovelluksissa. Yleisempiä painonappeja ovat tietenkin OK ja Cancel.



Javassa painonappi tehdään **Button**-luokan muodostimella. Muodostimia on kaksi kappaletta.

Button()	Painonappi, jolla ei ole nimeä
Button(String)	Painonappi, jolla on nimi

Kun painonappi on luotu, se pitää liittää container-olioon komponentiksi. Vasta tällöin painonappi tulee näkyviin. Liittäminen tapahtuu seuraavasti:

```
Button nappi = new Button (”Nappi1”);
add (nappi);
```

Painonapin nimi annetaan muodostimessa tai myöhemmin **`setLabel`**-funktiolla. Toinen tärkeä funktio on painonapin nimen lukufunktio **`getLabel`**. Painonappi sisältää lisäksi funktion tapahtumanimen asettamiseen ja lukemiseen.

getActionCommand()	Lukee tapahtumanimen
getLabel()	Lukee painonapin nimen
setActionCommand()	Asettaa tapahtumanimen
setLabel()	Asettaa painonapin nimen

Painonapin nimen voi muuttaa setLabel-funktiolla näin:

```
Button nappi = new Button (”Nappi1”)
nappi.setLabel (”Uusi nimi”);
```

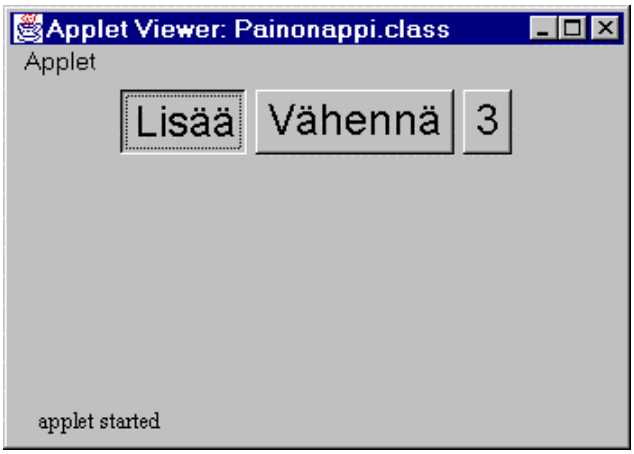
Painonapin painamisesta saadaan tietoa **ActionEvent**-luokalla ja rajapinnan ActionListener ja takaisinkutsuttavanfunktion **`actionPerformed`** avulla. Tällöin pitää komponenttiin asettaa kuuntelija funktiolla **`addActionListener`**. Haluttaessa tapahtumat voidaan käsitellä myös funktioilla processEvent (sallii tapahtumat) ja processActionEvent (takaisinkutsuttavafunktio). Komponentin tunnistamiseksi on meneteltävä seuraavasti:

1. Ensiksi sallitaan tapahtumat ja ylikirjoitetaan tapahtumasta kertova takaisinkutsuttavafunktio. Tarvittaessa getSource-funktiolla voidaan tapahtuman aiheuttaneen luokan esiintymä. Funktiota tarvitaan, jos useilla eri tyyppisillä komponenteilla on sama tapahtumanimi (tällöin instanceof-lauseella tutkitaan onko tapahtuma peräisin painonapista).
2. Painonapin nimeä verrataan **`getActionCommand`**-funktion palauttamaan nimeen. Vertaaminen tapahtuu equals-funktiolla. Mikäli merkkijonot ovat samansuuruiset, oikea painonappi on löytynyt.

Esimerkiksi painonapin ”Nappi1” tutkiminen tapahtuu yksinkertaisesti näin:

```
if (e.getActionCommand.equals(”Nappi1”));
 // oli nappi1
```

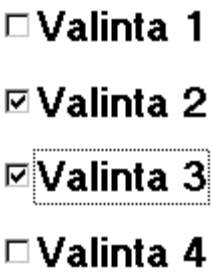
Seuraavassa on Java-ohjelma **Painonappi.java**, joka esittelee, kuinka painonappi luodaan, liitetään containeriin ja esitetään. Ohjelmassa tutustaan myös siihen, kuinka tapahtumia käsitellään action-funktiolla. Ohjelmassa tehdään 3 painonappia, joiden toiminta on seuraava: kaksi painonappia tekee laskutoimituksia luvulle (summaa ja vähentää). Kolmas painonappi esittää tämän luvun. Luvun nollaus tehdään painamalla tätä painonappia. Katso kuva.




```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Painonappi extends Applet implements
    ActionListener
{
 int luku=0;
 Button lisaa, vahenna, tulos;

 public void init()
 {
  add (lisaa = new Button("Lisää"));
  lisaa.addActionListener(this);
  add (vahenna = new Button("Vähennä"));
  vahenna.addActionListener(this);
  add (tulos = new Button("0"));
  tulos.addActionListener(this);
 }

 public void actionPerformed(ActionEvent e)
 {
  if (e.getActionCommand().equals("Lisää"))
   luku++;
  if (e.getActionCommand().equals("Vähennä"))
    luku--;
  if (e.getActionCommand().equals("Tulos"))
    luku=0;

   // Aseta luku painonapin tulos arvoksi
   tulos.setLabel(""+luku);
 }
}
```

9.2 Valintalaatikko

Valintalaatikko (*checkbox)* on nelikulmainen graafinen elementti, joka voidaan asettaa (rasti) tai nollata (tyhjä). Asettaminen ja nollaaminen tapahtuu vuorottaisesti. Piirtopinnalla saa olla samanaikaisesti useita valintalaatikoita, jotka kaikki voivat olla asetettuina tai nollattuina. Valintalaatikot ovat riippumattomia toisistaan.



Javassa valintalaatikko tehdään **CheckBox**-luokan muodostimella. Muodostimia on kolme kappaletta. Kahdella jälkimmäisellä muodostimella voidaan antaa valintalaatikolle nimi (sijoitetaan oikealle puolelle). Viimeisellä muodostimella voidaan lisäksi määrätä onko valintalaatikko asetettu (true) vai ei (false).

Checkbox()	Luo tyhjän valintalaatikon
Checkbox(String)	Luo nimellisen valintalaatikon
Checkbox(String, boolean)	Luo nimellisen valintalaatikon ja jolla on tietty tila

Kun valintalaatikko on luotu, se liitetään container-olioon komponentiksi. Vasta sen jälkeen valintalaatikko tulee näkyviin.

```
Checkbox vl = new Checkbox (”Valinta 1”);
add (vl);
```

Valintalaatikon nimi annetaan muodostimessa tai myöhemmin **`setLabel`**-funktiolla. Muita tärkeitä funktioita ovat valintalaatikon nimen lukufunktio **`getLabel`** sekä valintalaatikon tilan asetus- ja lukufunktiot. Valintalaatikko sisältää lisäksi funktion tapahtumanimen asettamiseen ja lukemiseen.

getActionCommand()	Lukee tapahtumanimen
getLabel()	Lukee valintalaatikon nimen

getState()	Lukee valintalaatikon tilan
getSelectedObjects()	Palauttaa kaikki valitut valintalaatikot
setActionCommand()	Asettaa tapahtumanimen
setLabel()	Asettaa valintalaatikon nimen
setState()	Asettaa valintalaatikon tilan

Valintalaatikon tilan voi lukea getState-funktiolla näin:

```
Checkbox vl = new Checkbox (”Valinta 1”);
int tila = vl.getState();
```

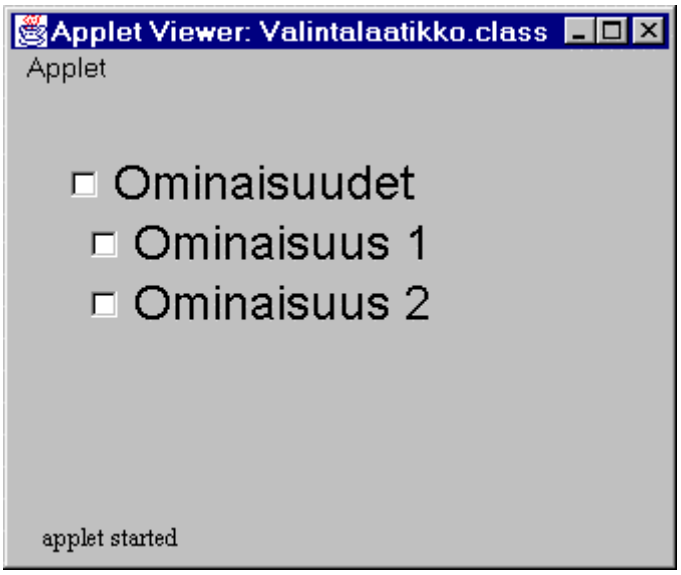
Valintalaatikon valinnasta saadaan tietoa **ItemEvent**-luokalla ja rajapinnan ItemListener ja takaisinkutsuttavan funktion **`itemStateChanged`** avulla. Tällöin pitää asettaa kuuntelija funktiolla **`addItemListener`**

1. Ensiksi sallitaan tapahtumat ja ylikirjoitetaan tapahtumasta kertova takaisinkutsuttavafunktio. Tarvittaessa getSource-funktiolla voidaan tapahtuman aiheuttaneen luokan esiintymä.
2. Valintalaatikon nimeä verrataan **`getItem`**-funktion palauttamaan nimeen. Vertaaminen tapahtuu equals-funktiolla. Mikäli merkkijonot ovat samansuuruiset, oikea valintalaatikko on löytynyt.

Esimerkiksi valintalaatikon ”Ominaisuudet” tutkiminen tapahtuu yksinkertaisesti näin:

```
if (e.getItem().equals("Ominaisuudet"))
    // valintalaatikko ominaisuudet
```

Java-ohjelma **Valintalaatikko.java** esittelee kuinka valintalaatikko luodaan, liitetään containeriin ja esitetään. Ohjelmassa tutustutaan myös siihen, kuinka tapahtumia käsitellään action-funktiolla. Ohjelmassa tehdään 3 valintalaatikkoa, joiden toiminta on seuraava: valintalaatikko ”Ominaisuudet” määrää ollessaan asetettuna, että kaksi muutakin valintalaatikkoa on asetettu. Jos valintalaatikko ”Ominaisuudet” on nollattu, kahden muun valintalaatikon tilaa ei ole määrätty, vaan ne voivat olla kumpikin eri tiloissa.




```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Valintalaatikko extends Applet implements
    ItemListener
{
 Checkbox kaikki,omin1,omin2;

 public void init()
 {
  setFont(new Font("Arial",Font.PLAIN,24));
  add (kaikki = new Checkbox("Ominaisuudet"));
  add (omin1 = new Checkbox("Ominaisuus 1"));
  add (omin2 = new Checkbox("Ominaisuus 2"));
  kaikki.addItemListener(this);
  omin1.addItemListener(this);
  omin2.addItemListener(this);
 }

 public void paint(Graphics g)
 {
  kaikki.setLocation(30,30);
  omin1.setLocation(40,70);
  omin2.setLocation(40,100);
 }
```

```
 public void itemStateChanged(ItemEvent e)
 {
   // Ominaisuudet -> kaikki checkboxit päälle
   if (e.getItem().equals("Ominaisuudet"))
   {
    omin1.setState(true);
    omin2.setState(true);
   }

   // Ominaisuus 1
   if (e.getItem().equals("Ominaisuus 1"))
   {
     if (kaikki.getState()==true)
      omin1.setState(true);
   }

   // Ominaisuus 2
   if (e.getItem().equals("Ominaisuus 2"))
   {
     if (kaikki.getState()==true)
       omin2.setState(true);
   }
  }
}
```

9.3 Radionappi

Radionappi (*radiobutton*) on pyöreä graafinen elementti, joka voidaan asettaa (ympyrä) tai nollata (tyhjä). Asettaminen ja nollaaminen tapahtuu vuorottaisesti. Radionappeja käytetään aina ryhmitettyinä. Se tarkoittaa sitä, että useita radionappeja kuuluu samaan ryhmään ja vain yksi ryhmän jäsenistä voi olla asetettu. Muut ovat siten nollattuja. Kun joku ryhmän jäsenistä asetetaan, muut ryhmän jäsenet nollautuvat automaattisesti.

Radionappi käyttää samaa luokkaa kuin edellä esitetty valintalaatikko, joten radionappi tehdään **Checkbox**-luokan muodostimella. Muodostimessa pitää olla parametri CheckboxGroup, joka ilmaisee radionappi ryhmän. Muodostimia on vain yksi.

Checkbox(String, CheckboxGroup, boolean)

Kun radionappi on luotu, se pitää liittää container-olioon komponentiksi. Vasta tällöin komponentti tulee näkyviin.

```
CheckboxGroup g = new CheckboxGroup();
Checkbox rd = Checkbox(”Nappi 1”, g, true);
add (rd);
```

Radionapin nimi annetaan muodostimessa tai myöhemmin **`setLabel`**-funktiolla. Muita tärkeitä funktioita ovat radionapin nimen lukufunktio **`getLabel`**, tilan asetus- ja lukufunktiot sekä ryhmän asetus- ja lukufunktiot **`setCheckboxGroup`** ja **`getCheckboxGroup`**.

getLabel()	Lukee radionapin nimen
setLabel()	Asettaa radionapin nimen
getState()	Lukee radionapin tilan
setState()	Asettaa radionapin tilan
getCheckboxGroup()	Lukee radionapin ryhmän
setCheckboxGroup	Asettaa radionapin ryhmän

Radionapin tilan voi lukea getState-funktiolla näin:

```
int tila = rd.getState();
```

Radionapin voi tehdä myös ajon aikana. Se tapahtuu määrittelemällä valintalaatikoille ryhmä (luomalla olio **CheckboxGroup**). Myös muodostin tarvitsee tämän olion, kuten edellä on jo huomattu. CheckboxGroup-olio alustetaan parametrittomalla muodostimella. Tämän jälkeen radionappeja lisätään ryhmään Checkbox-luokan **`setCheckboxGroup`**-funktiolla. Seuraavassa pieni esimerkki:

```
CheckboxGroup g = new CheckboxGroup();

// tee valintalaatikosta radionappi
Checkbox vl = new Checkbox (”Valinta 1”);
vl.setCheckBoxGroup(g);
```

```
// tee uusi radionappi
Checkbox rd = new Checkbox(”nimi”, g, false);
```

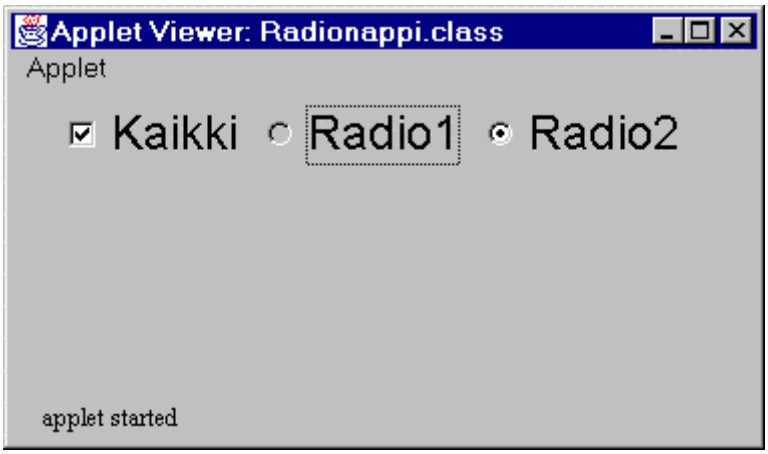
Radionapin valinnasta saadaan tietoa **ItemEvent**-luokalla ja rajapinnan ItemListener ja takaisinkutsuttavan funktion **`itemStateChanged`** avulla. Tällöin pitää asettaa kuuntelija funktiolla **`addItemListener`**.

1. Ensiksi sallitaan tapahtumat ja ylikirjoitetaan tapahtumasta kertova takaisinkutsuttavafunktio. Tarvittaessa getSource-funktiolla voidaan tapahtuman aiheuttaneen luokan esiintymä.
2. Radionapin nimeä verrataan **`getItem`**-funktion palauttamaan nimeen. Vertaaminen tapahtuu equals-funktiolla. Mikäli merkkijonot ovat samansuuruiset, oikea radionappi on löytynyt.

Esimerkiksi radionapin ”Radio2” tutkiminen tapahtuu yksinkertaisesti näin:

```
if (e.getItem().equals("Radio2"))
     // radionappi radio2
```

Java-ohjelma **RadioNappi.java** esittelee kuinka radionappi luodaan, liitetään containeriin ja esitetään. Ohjelmassa tutustaan myös siihen kuinka tapahtumia käsitellään action-funktiolla. Esimerkissä tehdään 2 radionappia ja 1 valintalaatikko. Elementtien toiminta on seuraava: valintalaatikko ”kaikki” on itsenäinen eli se voidaan asettaa ja sammuttaa vapaasti. Radionapit 1 ja 2 ovat samassa ryhmässä, joten vain toinen voi olla asetettu. Radionapin 2 painaminen asettaa myös valintalaatikon ”kaikki”.



```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Radionappi extends Applet implements
    ItemListener
{
 Checkbox kaikki;
 Checkbox radio1,radio2;
 CheckboxGroup group;

 public void init()
 {
  group = new CheckboxGroup();
  setFont(new Font("Arial",Font.PLAIN, 24));
  add (kaikki = new Checkbox("Kaikki"));
  add (radio1 = new Checkbox("Radio1",group,true));
  add (radio2 = new Checkbox("Radio2",group,false));
  kaikki.addItemListener(this);
  radio1.addItemListener(this);

  radio2.addItemListener(this);
 }

 public void itemStateChanged(ItemEvent e)
 {
   if (e.getItem().equals("Radio2"))
    kaikki.setState(true);
 }

}
```

9.4 Tekstikenttä

Tekstikenttä (*textfield*) on suorakulmainen graafinen elementti, jossa käyttäjä voi editoida yhtä tekstiriviä. Tekstikentässä on vakiona kaikki tekstinkäsittelyominaisuudet, kuten leikkaa- ja liimaa-toiminnot. Tekstikentän avulla voi esimerkiksi syöttää Java-ohjelmaan parametreja, tekstinä tai numeroina.

Tekstikenttä tehdään **TextField**-luokan muodostimella. Muodostimia on neljä kappaletta.

TextField()	Luo tyhjän tekstikentän
TextField(int)	Luo tyhjän tekstikentän, jolla on tietty pituus merkeissä
TextField(String)	Luo tekstikentän, jossa on merkkijono
TextField(String, int)	Luo tekstikentän, jossa on merkkijono ja tietty leveys merkeissä

Kun tekstikenttä on luotu, se pitää liittää container-olioon komponentiksi. Vasta sen jälkeen tekstikenttä tulee näkyviin.

```
TextField tf = new TextField ();
add (tf);
```

Tekstilaatikon ominaisuuksia ovat kyky editoida, leikata ja liimata tekstiä. Seuraavassa on esitelty funktioita, joilla suoritetaan tärkeitä tekstinkäsittelyyn tarkoitettuja toimenpiteitä. Funktiot löytyvät TextComponent-luokasta.

getText()	Lukee tekstin tekstikentästä
setText(String)	Asettaa tekstin tekstikenttään
getSelectedText()	Palauttaa valitun tekstin
getSelectionEnd()	Palauttaa valitun tekstin loppupaikan
getSelectionStart()	Palauttaa valitun tekstin alkupaikan
select(int, int)	Valitsee tekstin väliltä alku - loppu
selectAll()	Valitsee koko tekstin
isEditable()	Tosi, jos tekstikenttää voi editoida
setEditable(boolean)	Asettaa editoinnin (tosi = editointi päälle)
getColumns()	Palauttaa tekstikentän koon sarakkeissa

Esimerkiksi tekstikentän tekstin voi lukea getText-funktiolla näin:

```
TextField tf = new TextField ();
String teksti = tf.getText();
```

Tekstikentällä on vielä lisäksi yksi hyvä ominaisuus, joka liittyy salasanojen kirjoittamiseen. Salasanahan tunnetusti pidetään näkymättömissä muilta. Tekstikenttään kirjoitettavat merkit

voidaan korvata yhdellä merkillä, eli niin sanotulla kaiutusmerkillä. Tällöin näytölle ilmestyy kirjoitettaessa pelkästään kaiutusmerkkejä, esimerkiksi tähtiä. Kaiutukseen liittyvä funktio **`setEchoCharacter`** löytyy TextField-luokasta.

getEchoChar()	Lukee kaiutusmerkin
setEchoCharacter(char)	Asettaa kaiutusmerkin

Kaiutus voidaan asettaa päälle milloin vain setEchoCharacter-funktiolla:

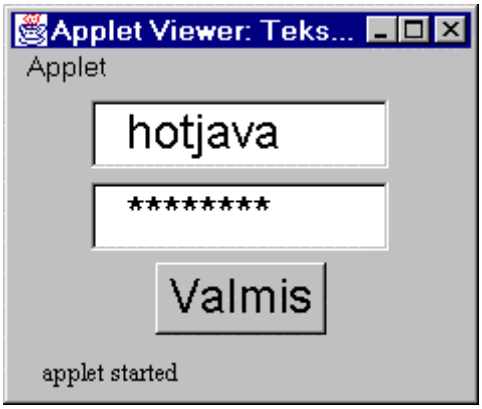
```
TextField tf = new TextField ();
tf. setEchoCharacter('*');
```

Tekstikentään kirjoitettaessa generoidaan tapahtuma **`textValueChanged`** eli tekstikentän sisältö on muuttunut. Tapahtumaluokka on **TextEvent** ja kuuntelija on TextEventListener. Kuuntelija asetetaan funktiolla **`addTextListener`**.

1. Ensiksi sallitaan tapahtumat ja ylikirjoitetaan tapahtumasta kertova takaisinkutsuttavafunktio.
2. Tekstikentän nimeä ei voi lukea tapahtumassa, vaan pelkästään siihen kirjoitettu teksti. GetSource-funktiolla voidaan lukea tekstikentän esiintymä, joka tapahtuman aiheutti.

Yleensä tekstikentän sisältö luetaan jonkun muun tapahtuman aikana. Tällöin tekstilaatikon pysyvän muuttujan avulla voidaan lukea tekstikentän sisältö kokonaan.

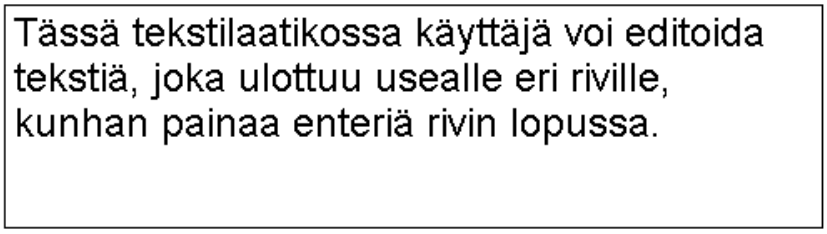
Java-ohjelma **Tekstikenttä.java** esittelee kuinka tekstikenttä luodaan, liitetään containeriin ja näytetään. Esimerkissä tehdään kaksi tekstikenttää ja yksi painonappi. Elementtien toiminta on seuraava: toinen tekstikenttä alustetaan sanalla ”hotjava”. Käyttäjän tehtävänä on kirjoittaa sana sama toiseen tekstikenttään kaiutusmerkeillä. Lopuksi painetaan painonappia ”Valmis”. Jos kirjoitus meni oikein, tulostuu hotjava-sanan tilalle teksti ”Oikein”.



```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Tekstikentta extends Applet implements
    ActionListener
{
 Button nappi;
 TextField lahde, kohde;
 String sana="hotjava";

 public void init()
 {
  add(lahde = new TextField(10));
  add(kohde = new TextField(10));
  add(nappi = new Button("Valmis"));
  lahde.setText(sana);
  kohde.setEchoChar('*');
  nappi.addActionListener(this);
 }

 public void actionPerformed(ActionEvent e)
 {
   if(lahde.getText().equals(kohde.getText()))
     lahde.setText("oikein");
 }

}
```

9.5 Tekstilaatikko

Tekstilaatikko (*TextArea*) on suorakulmainen graafinen elementti, jossa käyttäjä voi editoida tekstirivejä. Tekstilaatikossa on vakiona kaikki tekstinkäsittelyominaisuudet, kuten leikkaa- ja liimaa-toiminnot. Lisäksi siinä on vierityspalkit. Tekstilaatikon avulla Java-ohjelmaan voi syöttää parametreja, tekstinä tai numeroina. Tekstilaatikon avulla voidaan toteuttaa pienimuotoinen tekstieditori.



Tekstilaatikko tehdään TextArea-luokan muodostimella. Muodostimia on viisi kappaletta.

TextArea()	Luo tyhjän tekstilaatikon
TextArea(int, int)	Luo tyhjän tekstilaatikon, jolla on tietty korkeus ja leveys merkeissä
TextArea(String)	Luo tekstilaatikon, jossa on merkkijono
TextArea(String, int, int)	Luo tekstilaatikon, jolla on tietty korkeus ja leveys (merkeissä) sekä merkkijono

Viidennen muodostimen avulla määrätään kuinka vierityspalkit näkyvät. Muodostimen parametrit ovat samat kuin neljännen, mutta se sisältää lisäksi parametrin scrollbars.

TextArea(String, int, int, int)	Luo tekstilaatikon, jolla on tietty korkeus ja leveys (merkeissä), merkkijono sekä mahdollinen vierityspalkki

Parametrin mahdolliset arvot ovat:

SCROLLBARS_BOTH	Kummatkin vierityspalkit
SCROLLBARS_NONE	Ei vierityspalkkeja
SCROLLBARS_HORIZONTAL_ONLY	Vaakasuuntainen vierityspalkki
SCROLLBARS_VERTICAL_ONLY	Pystysuuntainen vierityspalkki

Kun tekstilaatikko on luotu, se pitää liittää container-olioon komponentiksi. Tämän jälkeen tekstilaatikko tulee näkyviin.

```
TextArea ta = new TextArea ();
add (ta);
```

Tekstilaatikon ominaisuuksia ovat kyky editoida, leikata ja liimata tekstiä. Seuraavassa on esitelty funktiot, joilla suoritetaan tärkeitä tekstinkäsittelyyn tarkoitettuja toimenpiteitä. Funktiot löytyvät TextComponent-luokasta.

getText()	Lukee tekstin tekstilaatikkoon
setText(String)	Asettaa tekstin tekstilaatikkoon
getSelectedText()	Palauttaa valitun tekstin
getSelectionEnd()	Palauttaa valitun tekstin loppupaikan
getSelectionStart()	Palauttaa valitun tekstin alkupaikan
select(int, int)	Valitsee tekstin väliltä alku - loppu
selectAll()	Valitsee koko tekstin
isEditable()	Tosi, jos tekstilaatikkoa voi editoida
setEditable(boolean)	Sallii/estää editoinnin

Esimerkiksi tekstilaatikon tekstin voi lukea getText-funktiolla näin:

```
TextArea ta = new TextArea();
String teksti = ta.getText();
```

Tekstilaatikolla on lisäksi omia funktioita, joita käytetään tekstin editointiin. Funktioilla **`append`** ja **`insert`** voi lisätä tekstiä, **`replaceRange`**-funktiolla voi vaihtaa tietyn merkkijonon toiseksi.

append	Lisää annetun tekstin tekstikentän loppuun
getColumns()	Lukee sarakkeiden määrän
getRows()	Lukee rivien määrän
getScrollbarVisibility()	Kertoo mitkä vierityspalkit on tekstilaatikolla
insert(String, int)	Lisää annetun tekstin tiettyyn kohtaan
replaceRange(String, int, int)	Vaihtaa tekstin tekstilaatikossa välillä alku - loppu
setColumns(int)	Asettaa sarakkeiden määrän

setRows(int)	Asettaa rivien määrän

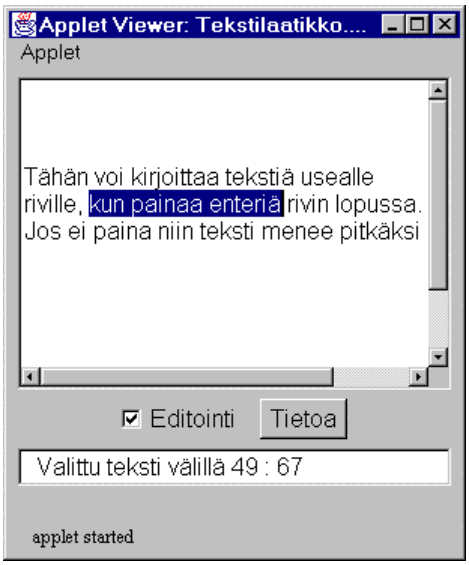
Esimerkiksi tekstin ”Moi!” lisääminen tekstilaatikon loppuun käy näin:

```
ta.append(”Moi!”);
```

Tekstilaatikkoon kirjoitettaessa generoidaan tapahtuma **`textValueChanged`** eli tekstilaatikon sisältö on muuttunut. Tapahtumaluokka on **TextEvent** ja kuuntelija on TextEventListener. Kuuntelija asetetaan funktiolla **`addTextListener`**.

1. Ensiksi sallitaan tapahtumat ja ylikirjoitetaan tapahtumasta kertova takaisinkutsuttavafunktio.
2. Tekstilaatikon nimeä ei voi lukea tapahtumassa, vaan pelkästään siihen kirjoitettu teksti. GetSource-funktiolla voidaan lukea sen tekstilaatikon esiintymä, joka tapahtuman aiheutti.

Java-ohjelma **Tekstilaatikko.java** esittelee, kuinka tekstilaatikko luodaan, liitetään containeriin ja näytetään. Esimerkissä tehdään yksi tekstilaatikko, yksi valintalaatikko sekä yksi painonappi ja yksi tekstikenttä. Tekstilaatikkoon voidaan kirjoittaa mitä tahansa tekstiä, editointi on päällä. Tämän määrää valintalaatikko Editointi. Painonappia painamalla saadaan tietoon valitun tekstin aloitus- ja loppupaikka. Tulosteeet ohjataan tekstikenttään. Katso kuva.




```
import java.awt.*;
import java.applet.*;
```

```
import java.awt.event.*;

public class Tekstilaatikko extends Applet implements
    ActionListener, ItemListener
{
 Button tietoa;
 TextArea ta;
 Checkbox edit;
 TextField tf;

 public void init()
 {
  resize(300,300);
  setFont(new Font("Arial",Font.PLAIN,16));
  add (ta = new TextArea(10,30));
  add (edit = new Checkbox("Editointi"));
  edit.setState(true);
  add (tietoa = new Button("Tietoa"));
  add (tf = new TextField(30));
  edit.addItemListener(this);
  tietoa.addActionListener(this);
 }

 public void itemStateChanged(ItemEvent e)
 {
  if (e.getItem().equals("Editointi"))
  {

   if (edit.getState())
    ta.setEditable(true);
   else
    ta.setEditable(false);
  }
 }

 public void actionPerformed(ActionEvent e)
 {
  if (e.getActionCommand().equals("Tietoa"))
   tf.setText("Valittu teksti välillä
    "+ta.getSelectionStart()+" : "+
    ta.getSelectionEnd());
 }
}
```

9.6 Vierityspalkki

Vierityspalkki (*scrollbar*) on graafinen elementti, joka ilmaisee elementin näkyvän osan riittämättömyyttä. Vierityspalkki aktivoituu, kun tietoa ei pystytä esittämään näkyvällä osalla. Javassa vierityspalkit ovat kuitenkin erillisiä elementtejä, joten niiden liittäminen johonkin tiettyyn elementtiin ei tapahdu yhdellä funktiokutsulla. Vierityspalkki on siirrettävä kohde-elementin viereen ja myös aktivoitava itse. Tärkeimmissä elementeissä, kuten tekstilaatikossa ja listassa, vierityspalkit ovat vakiona.

Javassa vierityspalkki tehdään Scrollbar-luokan muodostimella. Muodostimia on kolme kappaletta.

Scrollbar()	Muodostaa uuden vaakasuoran vierityspalkin
Scrollbar(int)	Muodostaa vierityspalkin, jolla on tietty suunta
Scrollbar(int, int, int, int, int)	Muodostaa vierityspalkin, jolla on tietty suunta, kohdistimen oletusarvo, sivun koko ja minimi- ja maksimiarvot

Vierityspalkin suunta määräytyy alla olevilla arvoilla:

HORIZONTAL	Vaakasuora vierityspalkki
VERTICAL	Pystysuora vierityspalkki

Kun vierityspalkki on luotu, se pitää liittää container-olioon komponentiksi. Vasta sen jälkeen vierityspalkki tulee näkyviin.

```
Scrollbar sh = new Scrollbar(Scrollbar.HORIZONTAL);
add (sh);
```

Vierityspalkin arvot voidaan asettaa muodostimessa tai myöhemmin funktioilla **`setValue`** tai **`setValues`**. Edellinen asettaa kohdistimen uuden sijainnin (jos rajoissa minimi/maksimi), jälkimmäinen asettaa kaikki uudet arvot (suunta, kohdistimen sijainti, sivun koko ja minimi- ja maksimiarvot).

setValue(int)	Asettaa kohdistimen sijainnin (jos rajoissa minimi/maksimi).
setValues(int,int,int,int)	Asettaa kaikki uudet arvot (suunta, kohdistimen sijainti, sivun koko ja minimi- ja maksimiarvot).

Esimerkiksi kohdistimen sijainti muutetaan näin:

```
sh.setValue(10);
```

Vierityspalkin tilaa voidaan tutkia seuraavilla funktioilla. Funktio **`getValue`** palauttaa kohdistimen sijainnin, **`getOrientation`** palauttaa vierityspalkin suunnan ja funktiot **`getMinimum`** ja **`getMaximum`** kertovat vierityspalkin minimi- ja maksimikoot.

getMaximum()	Palauttaa vierityspalkin maksimiarvon
getMinimum()	Palauttaa vierityspalkin minimiarvon
getOrientation()	Palauttaa vierityspalkin suunnan
getValue()	Palauttaa kohdistimen sijainnin

Kohdistimelle voidaan asettaa arvo, joka lisätään nykyiseen sijaintiin, kun mennään rivi/sivu (*unit/block*) ylöspäin tai alaspäin. Koska nämä arvot voidaan asettaa miksi tahansa, esimerkiksi rivin lisäys voi olla viisi riviä. Tästä syystä tulee nimitykset yksikkö (*unit*) ja lohko (*block*).

getUnitIncrement()	Lukee arvon rivi alaspäin/ ylöspäin
getBlockIncrement()	Lukee arvon sivu alaspäin/ ylöspäin
getUnitIncrement()	Asettaa arvon rivi alaspäin /ylöspäin
setBlockIncrement()	Asettaa arvon sivu alaspäin/ylöspäin

Vierityspalkin valinnasta on saadaan tietoa tapahtumafunktiolla **AdjustmentEvent**. Rajapinnan AdjustmentListener funktio **`adjustmentValueChanged`** reagoi vierityspalkin kohdistimen sijainnin muutokseen. Ennenkuin tapahtuma generoidaan, pitää asettaa kunntelija funktiolla **`addAdjustmentListener`**.

Tapahtuma voi olla yksi seuraavista:

BLOCK_DECREMENT	Kohdistin liikkui sivun alaspäin
BLOCK_INCREMENT	Kohdistin liikkui sivun alaspäin

TRACK	Kohdistin liikkui epämääräisen luvun verran ylöspäin tai alaspäin.
UNIT_DECREMENT	Kohdistin liikkui rivin alaspäin
UNIT_INCREMENT	Kohdistin liikkui rivin alaspäin

AdjustmentEvent-olion funktiolla **`getAdjustmentType`** luetaan tapahtuman tyyppi. Sitä verrataan yllä esitettyihin arvoihin. Funktiolla getAdjustable luetaan olio, joka tapahtuman aiheutti. (ScrollBar-luokka on periyttänyt Adjustable-rajapinnan.)

getAdjustable()	Palauttaa Adjustable-olion, joka aiheutti tapahtuman
getAdjustmentType()	Palauttaa tapahtuman tyypin
getValue()	Palauttaa vierityspalkin nykyisen arvon.

Esimerkiksi, jos tutkitaan onko vierityspalkin kohdistinta siirretty sivu alaspäin, menetellään näin:

```
public void adjustmentValueChanged(AdjustmentEvent e)
{
 if (e. getAdjustmentType() == e. BLOCK_DECREMENT)
  // on siirretty sivu alaspäin
}
```

Java-ohjelma **Vierityspalkki.java** esittelee kuinka vierityspalkki luodaan, liitetään containeriin ja näytetään. Esimerkissä tehdään kaksi vierityspalkkia, jotka liitetään appletin näyttöalueen reunoille. Vierityspalkeilla voi vierittää tekstiä ylä- ja alasuunnassa. Tekstin tulostaminen on toteutettu paint-funktiossa.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Vierityspalkki extends Applet implements
    AdjustmentListener
{
 Scrollbar sh;
 Scrollbar sv;
 String teksti = "Tässä on teksti, jota voi vierittää
vierintäpalkeilla";

 public void init()
 {
  resize(300,300);
  setFont(new Font("Arial",Font.PLAIN,24));
  sh = new Scrollbar(Scrollbar.HORIZONTAL,0,10,0,30);
  sv = new Scrollbar(Scrollbar.VERTICAL,10,10,0,30);
  sh.setUnitIncrement(1);
  sv.setUnitIncrement(1);
  sh.addAdjustmentListener(this);
  sv.addAdjustmentListener(this);
  add (sh);
  add (sv);
 }

 public void paint(Graphics g)
```

```
 {
  int xmax = getBounds().width;
  int ymax = getBounds().height;

  sh.setBounds(0, ymax-20, xmax-20, 20);
  sv.setBounds(xmax-20, 0 ,20, ymax-20);

  int merkki = sh.getValue();
  int korkeus = sv.getValue();
  g.drawChars(teksti.toCharArray(), merkki,
          teksti.length()-merkki, 20, korkeus*16);
 }

 public void adjustmentValueChanged(AdjustmentEvent e)
 {
   repaint();
 }
}
```

9.7 Nimiö

Nimiö (*label)* on suorakulmainen graafinen elementti, joka sisältää yksinkertaisen tekstin. Nimiön avulla voi esimerkiksi antaa käyttäjälle neuvoa, kuinka tulee toimia. Nimiön voi helposti sijoittaa jonkin muun elementin viereen. Kuvassa nimiö on sijoitettu radionappien yläpuolelle.

Nimiö tehdään Label-luokan muodostimella. Muodostimia on kolme kappaletta.

Label()	Luo tyhjän nimiön
Label(String)	Luo nimiön, jossa on teksti
Label(String, int)	Luo nimiön, jossa on teksti ja tasaus

Kun nimiö on luotu, se pitää liittää container-olioon komponentiksi. Vasta sen jälkeen nimiö tulee näkyviin. Liittäminen tapahtuu seuraavasti:

```
Label l = new Label (”Valitse tästä”);
add (l);
```

Nimiön nimi annetaan muodostimessa tai myöhemmin **`setText`**-funktiolla. Muita tärkeitä funktioita ovat nimiön tekstin lukufunktio **`getText`** sekä tasauksen asetusfunktio **`setAlignment`**.

getText()	Lukee nimiön tekstin
setText(String)	Asettaa nimiön tekstin
getAlignment()	Lukee nimiön tasauksen
setAlignment(int)	Asettaa nimiön tasauksen

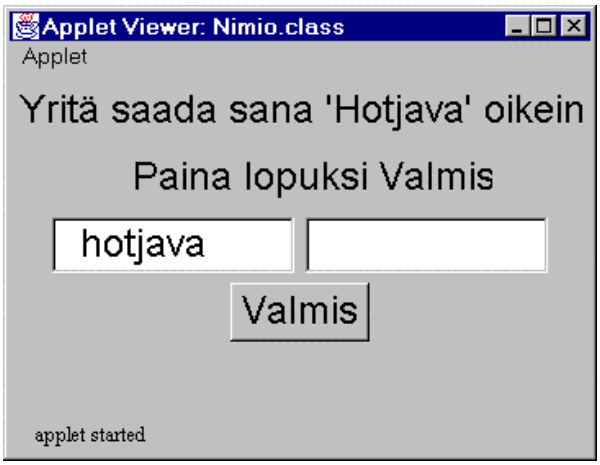
Nimiön tärkeä ominaisuus on tasaus, jonka avulla nimiön teksti voidaan sijoittaa esimerkiksi vasempaan reunaan. Tasaus asetetaan joko muodostimessa tai setAlignment-funktiolla. Tasaus määräytyy tasauslipulla, jotka ovat:

CENTER	Keskittää tekstin
LEFT	Tasaa tekstin vasempaan reunaan
RIGHT	Tasaa tekstin oikeaan reunaan

Esimerkiksi tasaus oikeaan reunaan tapahtuu näin:

```
l.setAlignment(l.RIGHT);
```

Java-ohjelma **Nimio.java** esittelee nimiön luontia, liittämistä ja näyttämistä. Ohjelmassa nimiö antaa tietoa ohjelman toiminnasta. Ohjelman runkona on käytetty tässä luvussa aiemmin esitettyä Java-ohjelmaa Tekstikentta.java.

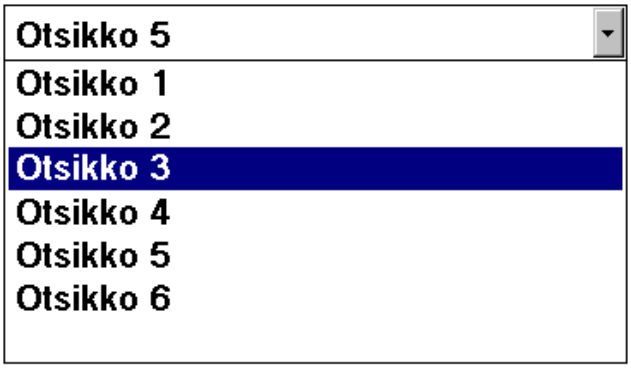



```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Nimio extends Applet implements
    ActionListener
{
 Button nappi;
 TextField lahde, kohde;
 String sana="hotjava";
 Label teksti;

 public void init()
 {
  setFont(new Font("Arial",Font.PLAIN, 24));
  teksti = new Label("Yritä saada sana 'Hotjava' oikein!
    ",Label.LEFT);
  add (teksti);
  teksti = new Label("Paina lopuksi Valmis");
  add (teksti);
  add(lahde = new TextField(10));
  add(kohde = new TextField(10));
  add(nappi = new Button("Valmis"));
  lahde.setText(sana);
  kohde.setEchoChar('*');
  nappi.addActionListener(this);
 }

 public void actionPerformed(ActionEvent e)
 {
   if(lahde.getText().equals(kohde.getText()))
     lahde.setText("oikein");
 }
```

```
}
```

9.8 Valintalista

Valintalista (*choice)* on suorakulmainen graafinen elementti, jossa on useita allekkaisia otsikoita (*items*). Valintalistasta näkyy vain yksi otsikko, kun se ei ole aktivoituna. Valintalista aukeaa, kun se aktivoidaan, ja kaikki siinä olevat otsikot tulevat näkyviin. Valintalistassa tulee aina olla vähintään yksi otsikko. Valintalista vastaa graafiselta tyyliltään pop-up-valikkoa. Valintalistaa kutsutaan Windowsissa nimellä Combobox.



Valintalista tehdään Choice-luokan muodostimella. Muodostimia on yksi.

Choice()	Luo tyhjän valintalistan

Kun valintalista on luotu, se pitää liittää container-olioon komponentiksi. Vasta tällöin valintalista tulee näkyviin.

```
Choice ch = new Choice ();
add (ch);
```

Valintalistan otsikot kirjoitetaan **`addItem`**-funktiolla, jos otsikot lisätään viimeiseksi. Otsikko lisätään tiettyyn paikkaan **`insert`**-funktiolla. Otsikoiden poistaminen tapahtuu remove-funktiolla. RemoveAll-funktiolla tyhjennetään koko valintalista yhdellä kertaa.

addItem(String)	Lisää uusi otsikko viimeiseksi
insert(String, int)	Lisää uusi otsikko tiettyyn paikkaan
remove(int)	Poistaa otsikon järjestysnumeron perusteella

remove(String)	Poistaa otsikon nimen perusteella
removeAll()	Poistaa kaikki otsikot

Otsikon lisääminen tapahtuu näin:

```
ch.addItem(”Otsikko 1”);
```

Otsikkojen lukeminen käy puolestaan **`getItem`**-funktiolla. Otsikoiden lukumäärä saadaan selville funktiolla **`getItemCount`**.

getItem(int)	Lukee otsikon nimen järjestysnumeron perusteella
getItemCount()	Palauttaa otsikoiden lukumäärän

Valintalistan otsikko valitaan ohjelmallisesti **`select`**-funktiolla. Käyttäjän valitsema otsikko luetaan funktiolla **`getSelectedItem`**.

getSelectedItem()	Palauttaa valitun otsikon nimen
getSelectedIndex()	Palauttaa valitun otsikon järjestysluvun
select(int)	Valitsee otsikon järjestysluvulla
select(String)	Valitsee otsikon nimellä

Otsikon ”Otsikko 1” valinta onnistuu näin:

```
ch.select(”Otsikko 1”);
```

Valintalistan valinnasta on saadaan tietoa tapahtumafunktiolla **ItemEvent**. Rajapinnan ItemListener funktio **`itemStateChanged`** reagoi otsikon valintaan. Ennenkuin tapahtuma generoidaan, pitää asettaa kuuntelija funktiolla **`addItemListener`**.

ItemEvent-luokan funktiolla **`getItem`** voidaan lukea mikä oli valitun otsikon nimi.

```
public void itemStateChanged(ItemEvent e)
{
if (e.getItem().equals("Laske otsikot"))
```

```
        // otsiko ”Laske otsikot” valittu
    }
```

Java-ohjelma **Valintalista.java** esittelee, kuinka valintalista luodaan, liitetään containeriin ja näytetään. Ohjelmassa opiskellaan myös tapahtumien käsittelyä action-funktiolla. Ohjelmassa tehdään yksi valintalista, jossa on neljä otsikkoa (katso kuva). Otsikoihin on ohjelmoitu erilaisia toimintoja, kuten otsikoiden lukumäärän laskenta.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Valintalista extends Applet implements
    ItemListener
{
 Choice c;

 public void init()
 {
  setFont(new Font("Arial", Font.PLAIN, 24));
  add (c = new Choice());
  c.addItem("Laske otsikot");
  c.addItem("Valitse viimeinen");
  c.addItem("Valitse minut");
  c.addItem("Lisää otsikko");
  c.addItemListener(this);
 }
```

```
 public void itemStateChanged(ItemEvent e)
 {
  if (e.getItem().equals("Laske otsikot"))
     showStatus("Otsikkoja: "+c.getItemCount());

  if (e.getItem().equals("Valitse viimeinen"))
     c.select(c.getItemCount()-1);

  if (e.getItem().equals("Valitse minut"))
     showStatus("Otsikkoni on: "+c.getSelectedItem());

  if (e.getItem().equals("Lisää otsikko"))
     c.addItem("Uusi otsikko");
 }

}
```

9.9 Lista

Lista (*list)*) on suorakulmainen graafinen elementti, jossa on allekkaisia otsikoita (*items*). Lista näkyy aina kokoinaisuudessaan riippumatta siitä, onko se aktivoitu vai ei. Listassa tulee olla vähintään yksi otsikko, mutta yleensä otsikoita on useampia. Kaikkien otsikoiden ei tarvitse olla kerralla näkyvissä. Tällöin vierityspalkkien avulla saadaan muut listan otsikot näkyviin.

Lista tehdään List-luokan muodostimella. Muodostimia on kolme kappaletta.

List()	Luo tyhjän listan
List(int)	Luo tyhjän listan, jolla on tietty määrä rivejä
List(int, boolean)	Luo tyhjän listan, jolla on tietty määrä rivejä sekä monivalintaisuus

Kun lista on luotu, se liitetään container-olioon komponentiksi. Tällöin lista tulee näkyviin. Liittäminen tapahtuu seuraavasti:

```
List l = new List ();
add (l);
```

Listan otsikot kirjoitetaan **addItem**-funktiolla, jos otsikot lisätään viimeiseksi. Otsikko lisätään tiettyyn paikkaan addItem-funktiolla, kun toiseksi parametriksi annetaan järjestysluku. Otsikoiden poistaminen tapahtuu **remove**-funktiolla. RemoveAll-funktiolla tyhjennetään koko lista yhdellä kertaa.

addItem(String)	Lisää uusi otsikko viimeiseksi
addItem(String, int)	Lisää uusi otsikko tiettyyn paikkaan
remove(int)	Poistaa otsikon järjestysnumeron perusteella
remove(String)	Poistaa otsikon nimen perusteella
removeAll()	Poistaa kaikki otsikot

Otsikon lisääminen tapahtuu näin:

```
l.addItem(”Otsikko 1”);
```

Otsikkojen lukeminen käy tapahtuu **getItem**-funktiolla. Otsikoiden lukumäärä saadaan selville funktiolla **getItemCount**.

getItem(int)	Lukee otsikon nimen järjestysnumeron perusteella
getItemCount()	Palauttaa otsikoiden lukumäärän

Listan otsikko valitaan ohjelmallisesti **select**-funktiolla. Käyttäjän valitsema otsikko luetaan funktiolla **getSelectedItem**. Funktiolla **isIndexSelected** voidaan testata onko jokin tietty otsikko valittu.

getSelectedItem()	Palauttaa valitun otsikon nimen
getSelectedIndex()	Palauttaa valitun otsikon järjestysluvun
isIndexSelected(int)	Testaa onko otsikko valittu
select(int)	Valitsee otsikon järjestysluvulla

select(String)	Valitsee otsikon nimellä
deselect(int)	Poistaa otsikon valinnan

Otsikon ”Otsikko 1” valinta onnistuu näin:

```
ch.select(”Otsikko 1”);
```

Listalla on vielä eräs hyvä ominaisuus joka valintalistalta puuttuu. Se on monivalintaisuus. Se tarkoittaa sitä, että useita otsikoita voidaan valita samalla kertaa. Monivalintaisuutta varten listalla on seuraavat funktiot:

isMultipleMode()	Palauttaa arvon tosi, jos monivalintaisuus on sallittu
setMultipleMode(boolean)	Asettaa monivalintaisuuden
getSelectedIndexes()	Palauttaa valittujen otsikoiden järjestysluvut
getSelectedItems()	Palauttaa valittujen otsikoiden nimet

Listan valinnasta on saadaan tietoa tapahtumafunktiolla **ItemEvent**. Rajapinnan ItemListener funktio **`itemStateChanged`** reagoi otsikon valintaan. Ennenkuin tapahtuma generoidaan, pitää asettaa kuuntelija funktiolla **`addItemListener`**.

ItemEvent-luokan funktiolla **`getItem`** voidaan lukea mikä oli valitun otsikon järjestysnumeron (huomaa, että valintalista palautti otsikon nimen, ei järjestysnumeroa). Liittämällä ActionListener-rajapinta listaan, saadaan ActionEvent-luokan tapahtumafunktiolla ActionPerformed ja funktiolla getActionCommand luettua otsikon nimen.

```
if (e.getActionCommand().equals(”otsikko”))
    // valittiin otsikko
```

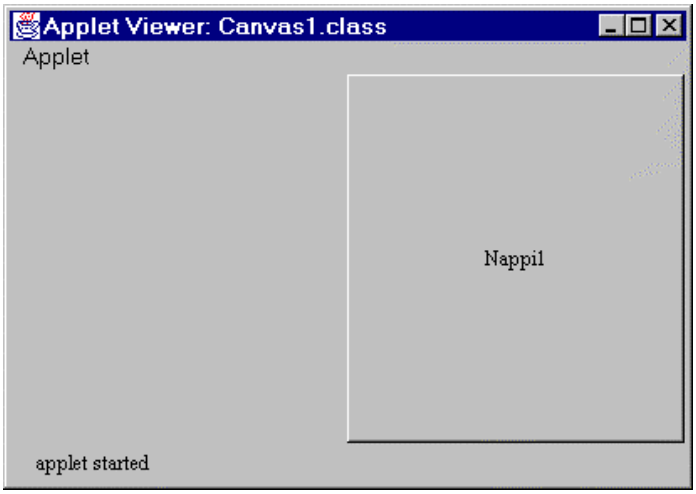
Java-ohjelma **Lista.java** esittelee kuinka valintalista luodaan, liitetään containeriin ja näytetään. Ohjelmassa tehdään yksi lista, jossa on neljä otsikkoa (katso kuva). Otsikoihin on ohjelmoitu erilaisia toimintoja, kuten otsikoiden lukumäärän laskenta.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Lista extends Applet implements
ActionListener
{
 List l;

 public void init()
 {
  setFont(new Font("Arial",Font.PLAIN,24));
  add (l = new List());
  l.addItem("Laske otsikot");
  l.addItem("Valitse minut");
  l.addItem("Lisää otsikko");
  l.addItem("Poista otsikko");
  l.addActionListener(this);
 }

 public void actionPerformed(ActionEvent e)
 {
   if (e.getActionCommand().equals("Laske otsikot"))
    showStatus("Otsikkoja: "+l.getItemCount());

   if (e.getActionCommand().equals("Valitse minut"))
    showStatus("Valitsit otsikon: "+l.getSelectedItem());

   if (e.getActionCommand().equals("Lisää otsikko"))
    l.addItem("Uusi otsikko");

   if (e.getActionCommand().equals("Poista otsikko"))
    l.delItem(l.getItemCount()-1);
 }
```

```
}
```

9.10 Canvas-elementti piirtopintana

Canvas on elementti, joka toimii piirtopintana. Toisin kuin paneeli, Canvas ei voi sisältää muita elementtejä. Canvas-elementin käyttötarkoitus onkin lähinnä toimia yksinkertaisena piirtopintana, johon voidaan kohdistaa graafisia toimintoja Graphics-luokan funktioilla. Tämä käsittää erilaisten grafiikan, tekstin sekä kuvien tulostamisen. Canvas-luokka sisältääkin vain yhden funktion - se on tietenkin paint. Canvas-elementtiä voidaan myös käyttää tyhjänä elementtinä muiden elementtien joukossa.



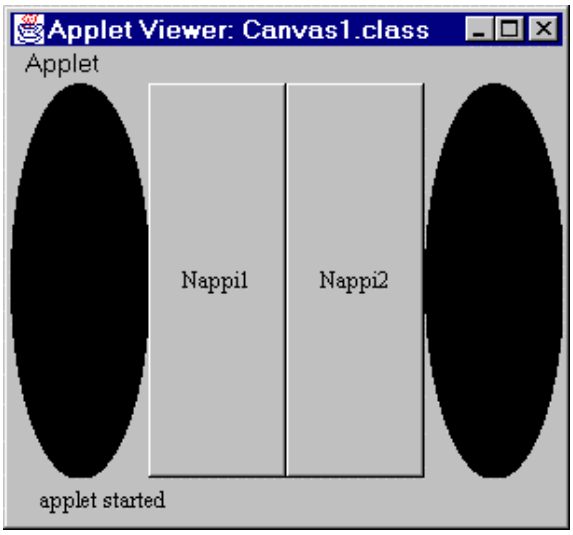
Canvas-elementti muodostetaan Canvas-luokan ainoalla muodostimella, joka on parametriton.

Canvas()	Luo Canvas-elementin

Kun elementti on luotu, se liitetään containeriin add-funktiolla.

```
Canvas cn = new Canvas();
add (cn);
```

Java-ohjelma **Canvas1.java** esittelee Canvas-elementin luontia, liittämistä sekä paint-funktion ylikirjoittamista. Ohjelmassa luodaan toinen luokka nimeltä Canvas1, joka edustaa ellipsin piirtävää Canvas-elementtiä. Katso kuva.




```
import java.awt.*;
import java.applet.*;

public class Canvas1 extends Applet
{
 public void init()
 {
  setLayout(new GridLayout(1,4));
  add (new Canvas2());
  add (new Button("Nappi1"));
  add (new Button("Nappi2"));
  add (new Canvas2());
 }
}

class Canvas2 extends Canvas
{
 public void paint(Graphics g)
 {
  g.fillOval(0,0, getBounds().width, getBounds().height);
 }
}
```

9.11 Elementtien sijoittelu

Elementtien sijoittaminen piirto-alueelle on yksi tärkeimpiä toimnepiteitä ohjelman käyttöliittymän teossa. Ei ole mitään järkeä sijoittaa elementtejä mihin sattuu. Elementit pitää sijoittaa järkevästi siten, että ohjelmalle syntyy hyvä ulkoasu.

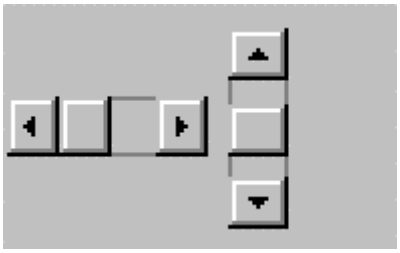
Elementtien sijoittamiseen on kaksi vaihtoehtoa. Ensimmäinen ja helpoin on käyttää Component-luokan funktioita, jotka vaikuttavat elementtien sijaintiin ja kokoon. Tällöin elementit voidaan sijoittaa pikselin tarkkuudella. Tällaisia funktioita ovat muun muassa **`setLocation`**, **`setBounds`** ja **`setSize`**.

setLocation(int, int)	Siirtää elementin paikkaan x, y
setBounds(int, int, int, int)	Siirtää elementin paikkaan x, y ja muuttaa elementin korkeuden ja leveyden
setSize(int, int)	Muuttaa elementin kokoa

Esimerkiksi vierityspalkkien yhteydessä tarvittiin funktiota setBounds muuttamaan vierityspalkkien sijantia ja kokoa. Sijainti muutettiin siten, että vierityspalkit tulivat appletin reunoille. Kokoa muutettiin siten, että vierityspalkit täyttivät koko reunan. SetBounds-funktiota käytettiin paint-funktiossa seuraavasti:

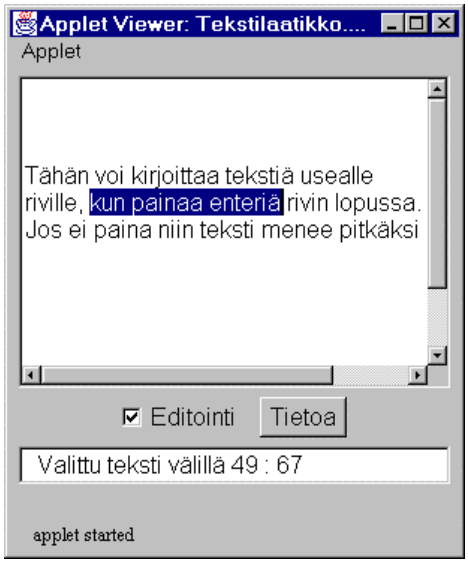
```
sh.setBounds(0, ymax-20, xmax-20, 20);
sv.setBounds(xmax-20, 0 ,20, ymax-20);
```

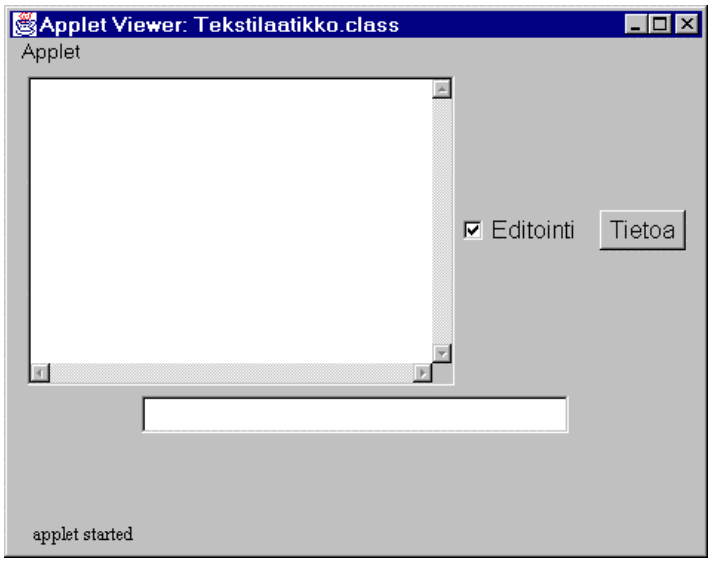
Ellei näin olisi tehty, vierityspalkit olisivat olleet minimaalisen pieniä sekä sijoittuneina layoutin mukaan (katso alla oleva kuva).



Toinen tapa, kuten arvata saattaa, on tietenkin layout. Layouttien etu on se, että elementit sijoittuvat näyttöalueen koon mukaan. Jos näyttöalueen koko muuttuu, samalla myös elementtien sijoittelu muuttuu. Koon kasvaessa myös elementtien koko voi kasvaa (riippuu tietenkin layoutista) ja päinvastoin.

Kun ohjelmoija löytää sopivat mitat appletille, elementit sijoittuvat näytölle hyvin. Otetaan lyhyt esimerkki: Java-ohjelmassa Tekstilaatikko.java sopivat mitat ovat 300 * 300 (kuvassa vasemmalla). Jos kokoa muuttaa, ohjelman ulkoasu saattaa näyttää aivan hullulta (kuvassa oikealla). Onneksi näyttöalueen koon muutoksen voi estää, joten lopputulos saadaan näyttämään hyvältä.





Ohjelmassa ei tietenkään tarvitse käyttää samaa layouttia koko ajan. Tapauksesta riippuen layoutteja kannattaa lisätä ohjelmaan useampia.
Layouttien haitaksi voidaan laskea niiden vähyys - oikeanlaista ei tahdo löytää millään. Layouteista paras on gridbag, mutta sen käyttö on myös kaikista hankalinta - lopputuloksen näkee vasta, kun kaikki elementit ovat kasassa.

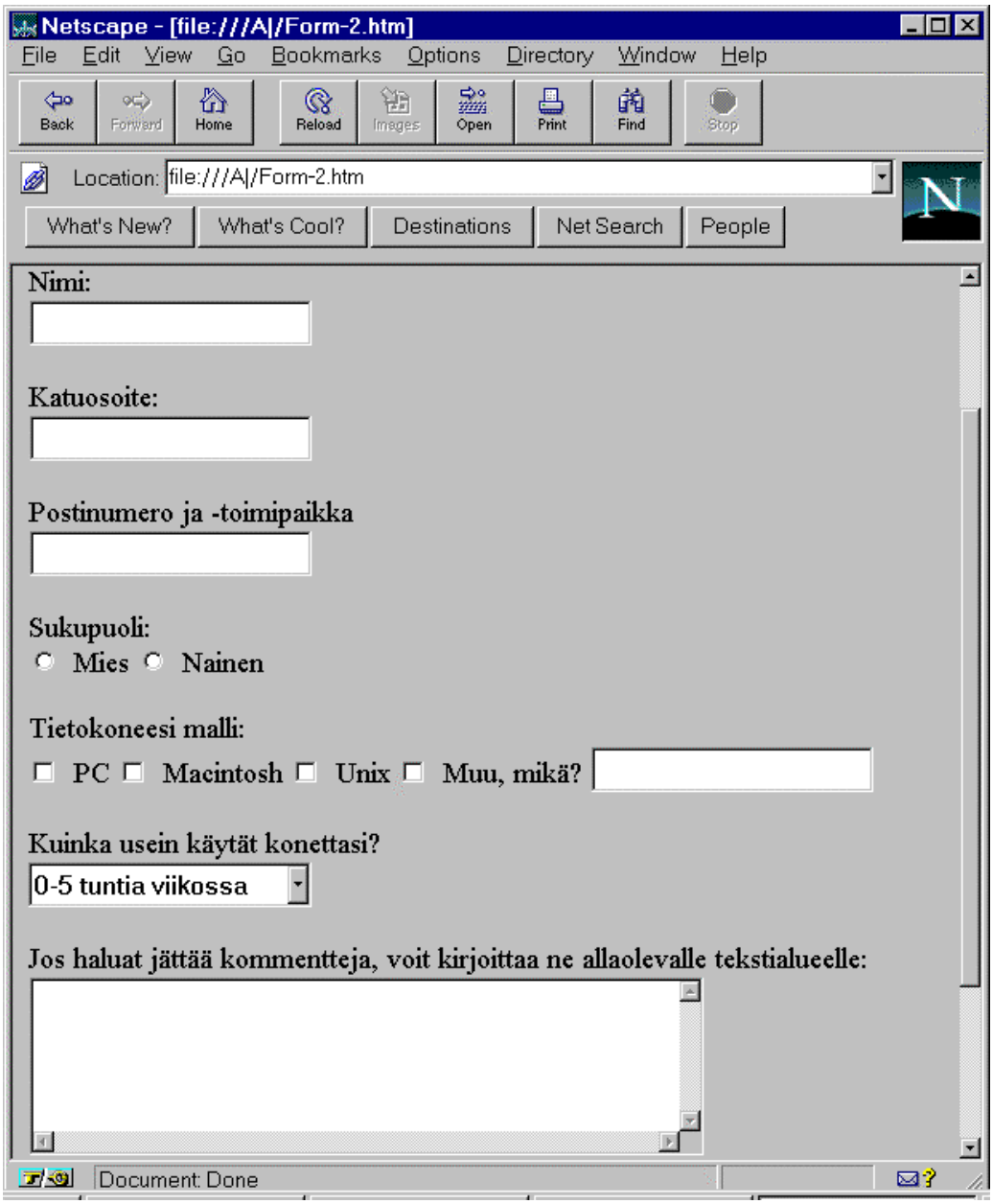
9.12 Lomakkeet

Lomakkeet on WWW:ssä tarkoitettu interaktiivisivien osien toteutukseen. Tällaisia ovat muun muassa kyselyt ja kaavakkeet (tilaus, rekisteröinti). Lomake ei ole erillinen elementti, vaan se on yhdistelmä graafisista elementeistä, kuten painonapeista, tekstikentistä ja listoista.

Lomakkeiden teossa komponenttien sijoittelu layoutin kanssa saattaa olla helpompaa kuin Component-luokan funktioilla. Yleensä lomake on toteutettu useista allekkaisista elementeistä. Esimerkiksi seuraavan sivun kuvassa tämä näkyy erittäin hyvin. Vain muutamat elementit, kuten radionapit ja valintalaatikot ovat rinnakkain.

Kuvassa näkyvät tekstit, kuten Nimi, on toteutettu yksinkertaisesti nimiöllä. Nimiö onkin on paras tapa antaa informaatiota käyttäjälle elementtien tarkoituksesta.

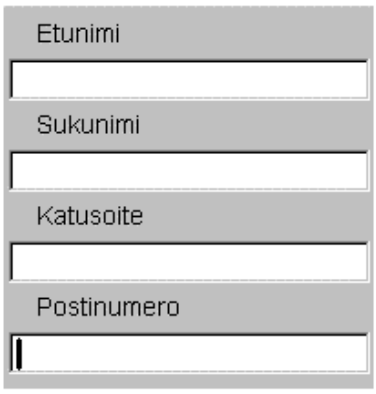
Lomakkeet voidaan toteuttaa myös HTML:n avulla, mutta ilman JavaScript-kieltä niitä ei voi hyödyntää Java-apleteissa. Apletit eivät tiedä eivätkä välitä siitä mitä HTML-sivu sisältää. HTML-elementtien, JavaScriptin ja Javan yhdistämistä käsitellään syvällisemmin kirjan Opeta itsellesi Java-ohjelmointi kolmannessa painoksessa sekä syksyllä julkaistavassa kirjassa Tehokäyttäjän opas: WWW-ohjelmointi.



HTML-kieli yhdessä CGI:n kanssa tarjoavat monipuoliset lomakkeiden käsittelytoiminnot. HTML-kielellä tehdään lomakkeeseen elementit ja sitten tulokset ohjataan CGI-ohjelmalle, joka vastaa tietojenkäsittelystä. Lopuksi vielä palautetaan vastauksena käyttäjälle, jokin viesti (hyväksytty tai hylätty). Tästä menettelytavasta johtuen HTML ja CGI ei tarjoa välitöntä interaktiivisuutta, koska vasta CGI-ohjelman tuloksesta voi päätellä, hyväksyttiinkö tiedot oikeiksi. Esimerkiksi jos lomakkeessa pyydetään käyttäjän osoitetietoja, postinumeron paikalle voi kirjoittaa tekstiä, vaikka siinä pitäisi olla numeroita.

Javassa välitön interaktiivisuus, joka reagoi postinumeron oikeellisuuteen, on mahdollista toteuttaa melko helposti. Tutkitaan takaisinkutsuttavassa tapahtumafunktiossa sisältääkö tekstikenttä pelkästään numeroita. Jos ei sisällä niin tekstikentän sisältöä ei hyväksytä. Ilmoitus tästä voidaan ohjelmoida tyhjentämällä tekstikenttä.

Javan avulla voidaan mennä myös paljon pidemmälle. Ohjelmoidaan applettiin kaikki mahdolliset postinumerot, jolloin voidaan heti tarkastaa, oliko annettu postinumero oikea.

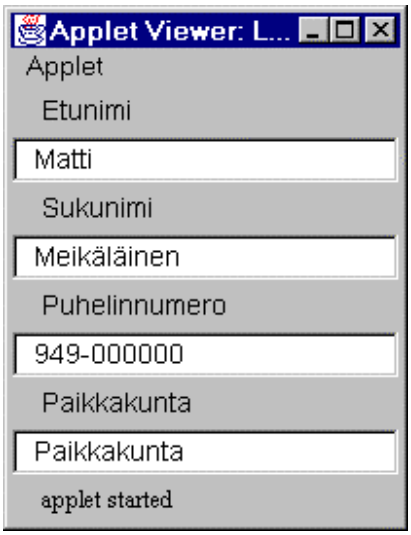


Java-ohjelma **Lomake.java** toteuttaa yllä esitetyn kuvan mukaisen appletin, jossa on postinumeron tarkastus (=kentässä ei hyväksytä kirjaimia).

```
import java.awt.*;
import java.applet.*;
import java.net.*;
import java.awt.event.*;

public class Lomake extends Applet implements
    TextListener
{
```

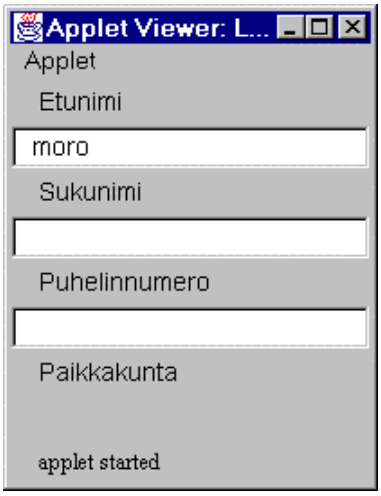
```
 TextField etunimi, sukunimi, katuosoite, postinro;

 public void init()
 {
  setFont(new Font("Arial", Font.PLAIN, 14));
  setLayout(new GridLayout(8,1));
  add (new Label("Etunimi"));
  add (etunimi = new TextField(20));
  add (new Label("Sukunimi"));
  add (sukunimi = new TextField(20));
  add (new Label("Katusoite"));
  add (katuosoite = new TextField(20));
  add (new Label("Postinumero"));
  add (postinro = new TextField(5));
  postinro.addTextListener(this);
 }

 public void textValueChanged(TextEvent e)
 {
   String teksti = postinro.getText();

   for (int i = 0 ; i<teksti.length(); i++)
    if (!Character.isDigit(teksti.charAt(i)))
      postinro.setText("");
 }
}
```

Välittömän interaktiivisuuden etu on myös se, että elementit voidaan kytkeä pois päältä ellei niitä tarvita. Esimerkiksi kaavakkeessa voi olla kohta, johon kirjoitetaan puhelinnumero. Puhelinnumeron perusteella pystytään päättelemään asuinpaikkakunta. Mutta jos kirjoitetaan matkapuhelinnumero, paikkakuntaa ei voi päätellä. Tällöin pitäisi täyttää erikseen kohta, joka kertoo asuinpaikkakunnan (katso kuva).



Component-luokasta löytyy funktiot, joilla elementtien käyttö voidaan estää ja sallia. Samasta luokasta löytyy funktiot myös elementtien piilottamiseen ja näyttämiseen. Yllä olevassa tapauksessa paikkakunnan ilmoittava elementti voidaan pitää näkyvissä. Mutta mikäli puhelinnumero ei olisikaan matkapuhelimeen, elementti voitaisiin piilottaa.

setEnabled(false)	Estää elementin käytön
setEnabled(true)	Sallii elementin käytön
setVisible(false)	Piilottaa elementin
setVisible(true)	Näyttää elementin



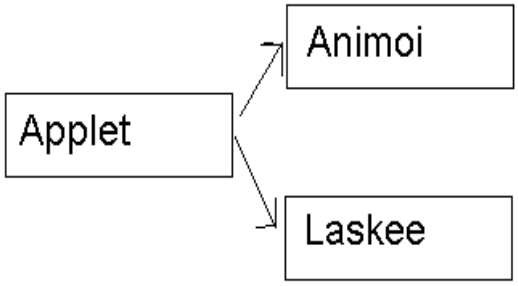
10. Säikeet Javassa

Säie-ohjelmointi on uusinta uutta moniajokäyttöjärjestelmissä, ja siitä on erityisesti hyötyä moniprosessoripohjaisissa tietokoneissa. Myös Java-kieli tukee säie-ohjelmointia, joten sitä kannattaa hyödyntää omissa ohjelmissa. Tämän luvun aikana opetellaan säie-ohjelmoinnin perusteet, ja kerrotaan missä säikeitä kannattaa käyttää.

10.1 Mitä ovat säikeet?

Sana säie saattaa kuulostaa monelle aika oudolta. Mitä tekemistä Java-ohjelmoinnilla on säikeiden kanssa? Paljonkin, sillä säikeitä tarvitaan esimerkiksi animaation yhteydessä.

Säie (*thread*) on ohjelman käynnistämä prosesssi. Javassa säikeitä voisi nimittää myös Thread-prosessiksi. Kun moniajoympäristössä puhutaan usean samanaikaisen ohjelman ajamisesta, säikeiden yhteydessä puhutaan useista prosesseista, jotka jokin tietty ohjelma on aktivoinut.



Prosessi on itsenäisesti suoriutuva ohjelman osa. Se voi olla esimerkiksi animaattori (kuten kuvassa) tai laskuri, joka laskee appletin voimassaoloaikaa. Yksi appletti voi avata useita säikeitä samanaikaisesti suoritettavasti. Säikeitä ajetaan siis rinnakkaisesti.

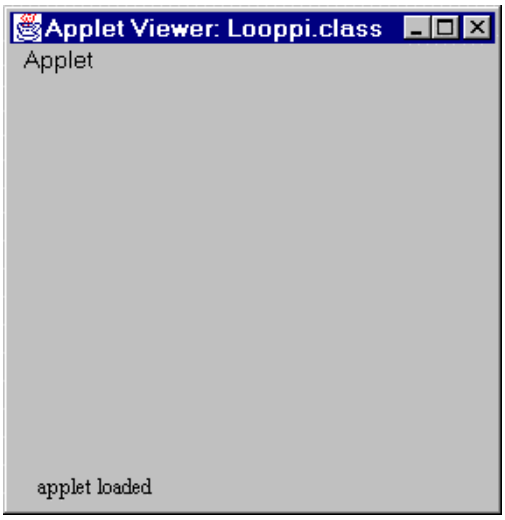
10.2 Miksi säikeitä tarvitaan?

Yksi säikeiden käyttökohde tuli jo kerrottua edellä: animaatio. Animaatiossa siksi, että animaation tuottaminen kuluttaa paljon aikaa. Kun aikaa kuluttavat tehtävät suoriutuvat itsenäisissä säikeissä, järjestelmä pystyy suorittamaan toimenpiteensä häiriintymättä. Varsinkin Java-ympäristössä kaikkien takaisinkutsuttavienfunktioiden ohjelmakoodi on oltava säikeessä, mikäli ne kuluttavat paljon aikaa.

Monisäikeisyyden ansiosta voidaan suorittaa monta applettia samanaikaisesti. Tietenkin applettien suorittaminen hidastuu, mutta kuitenkin niitä ajetaan.

Start-funktioon ei kannata laittaa aikaa kuluttavaa silmukkaa, vaikka aluksi saattaa tuntua siltä, että applet toimii niin kuin pitääkin. Tällöin applet valitettavasti tekee vastarintaa ja estää järjestelmän toiminnan. Esimerkiksi tärkeän paint-funktion kutsuminen estyy, joten piirtopinnalle ei voi piirtää. Kun aikaa kuluttava osa siirretään säikeeseen, niin järjestelmä toimii normaalisti. Java-ohjelma **Looppi.java** esittelee, kuinka aikaa kuluttava silmukka haittaa järjestelmän toimintaa.

Ohjelman pitäisi tulostaa kellonaika ikkunaan. Ohjelma on toteutettu niin, että se kutsuu repaint-funktiota ikuisessa silmukassa (katso seuraavan esimerkin Looppi.java init-funktio). Tuloksena on tyhjä ikkuna, koska paint-funktiota ei pystytä kutsumaan.




```
import java.awt.*;
import java.applet.Applet;
import java.util.*;
import java.text.*;

public class Looppi extends Applet
{
 public void init()
 {
  while (true)
   repaint();
 }

 public void paint(Graphics g)
 {
  Calendar cal = Calendar.getInstance();

  g.setFont(new Font("Arial",Font.PLAIN,48));
  g.drawString(DateFormat.getTimeInstance().format(new
Date()), 10, 60);
 }
}
```

10.3 Säie-ohjelmointi Thread-luokalla

Tähän mennessä appletit on ohjelmoitu niin, että init-funktiossa on suoritettu alustaminen ja on kutsuttu tarvittavia funktioita. Luvussa 5 esiteltyä start-funktiota ei ole vielä tarvittu. Säikeiden yhteydessä sitä tullaan tarvitsemaan. Sen lisäksi tarvitaan vielä run-funktiotakin, josta ei ole vielä kerrottu. Säie-ohjelmointiin on kaksi eri tapaa, Thread-luokka tai Runnable-rajapinta. Tutustutaan aluksi Thread-luokkaan. Luokka sisältyy pakettiin **java.lang**, joten sitä ei tarvitse erikseen ladata import-käskyllä.

```
java.lang.Object
  |
  +----java.lang.Thread
```

Start-funktion aikana on hyvä käynnistää ohjelman tarvitsema säie, koska ohjelman alustaminen on jo suoritettu init-funktion aikana. Oikeastaan start-funktio on tavallaan tehtykin säikeen aloittamista varten. Säie aloitetaan luomalla oma prosessi. Koodissa prosessi on luokka, joka on periytynyt Thread-luokasta. Hyvä tapa on tarkastaa, että säie ei ole toiminnassa ennen kuin se alustetaan (ellei välttämättä haluta useita säikeitä).

```
prosessi p;

public void start()
{
 if (p != null) {
  p = new prosessi();
  p.start();
 }
}
```

Säikeen suorittaminen aloitetaan **`run`**-funktiosta, kun kutsutaan prosessiluokan **`start`**-funktiota. Prosessiluokkaan tarvitsee koodata run-funktio, joka suorittaa tarvittavat tehtävät.

```
public void run()
{
  // tässä suoritetaan ohjelmaa
}
```

Säie on elossa niin kauan kunnes run-funktiosta poistutaan, eli prosessi loppuu, kun run-funktio loppuu. Stop-funktiossa kannattaa varmistaa säikeen lopettaminen kutsumalla Thread-luokan **`stop`**-funktiota.

```
public void stop()
{
 if (p != null)
```

```
  p.stop()
 p = null;
}
```

Haittana Thread-luokan käytössä on se, että prosessiluokka joutuu sisällyttämään Thread-luokan. Koska ei ole moniperiytyvyyttä, samanaikainen periytyminen prosessiluokkaan ei ole mahdollista Applet- ja Thread-luokista. Tästä syystä prosessin aloittamiseen tarvitaan kaksi erillistä luokkaa: Applet ja Thread. Seuraavassa on esimerkki säikeen aloittamisesta Thread-luokan avulla. Samalla se korjaa tämän luvun ensimmäisen esimerkkiohjelman puutteet.

```
// Pääohjelma
class Ohjelma extends Applet
{
 public void  start()
 {
  prosessi p  = new prosessi();
  p.start();
 }
}

// prosessi (erillinen luokka)
class prosessi extends Thread
{
 public void run()   // ylikirjoita run
 {
  // tähän tulee oma ohjelma
 }
}
```

10.4 Runnable-rajapinta

Runnable-rajapinta on toinen tapa alustaa säie. Kummassakin tavassa idea on sama: ylikirjoitetaan takaisinkutsuttava run-funktio, jossa on säikeen varsinainen ohjelmakoodi.

Runnable-rajapinta on suunniteltu antamaan yhteinen käsittelytapa kaikille appleteille, jotka haluavat suorittaa koodia ollessaan aktiivisia. Myös edellä kerrottu Thread-luokka käyttää Runnable-rajapintaa hyväkseen. Jotta rajapinta saadaan käyttöön, pitää se lisätä implements-sanalla luokan määrittelyyn.

```
class Ohjelma extends Applet implements Runnable
```

Start-funktion aikana on hyvä käynnistää ohjelman tarvitsema säie, koska ohjelman alustaminen on jo suoritettu init-funktion aikana. Säie aloitetaan luomalla oma prosessi. Koodissa prosessi on Thread-luokan olio. Kahta erillistä luokkaa ei tarvita! Hyvä tapa on tarkastaa, että säie ei ole toimissa ennen kuin se alustetaan.

```
Thread prosessi;

public void  start()
{
 if (prosessi == null)
  prosessi = new Thread(this);
  prosessi.start();
}
}
```

Säikeen suorittaminen aloitetaan **run**-funktiosta, kun kutsutaan prosessiluokan **start**-funktiota. Run-funktio ylikirjoitetaan applet-luokkaan.

```
public void run()   // ylikirjoita run-metodi
 {
  //  tähän tulee oma ohjelmakoodi
 }
}
```

Säie on elossa niin kauan kunnes run-funktiosta poistutaan, eli prosessi loppuu, kun run-funktio loppuu. Haittana tässä tavassa on se, että run-funktio on kaikille säikeille sama. Jos halutaan aktivoida useita säikeitä, tulee käyttää erillisiä luokkia, joissa rajapintana on Runnable. Seuraavassa on esimerkki säikeen aloittamisesta Runnable-rajapinnan avulla.

```
// Pääohjelma & prosessi
Class Ohjelma extends Applet implements Runnable
{
 public void  start() // Aloita Threadin suoritus
 {
  Thread th = new Thread(this);
  th.start();   // kutsuu run-metodia
 }

 public void run()  // ylikirjoita run-metodi
 {
  //  tähän tulee oma ohjelmakoodi
 }
}
```

10.5 Säikeen tilan ohjaus

Thread-luokka sisältää useita funktioita, joilla voidaan ohjata säiettä. Yksi hyvä funktio on **`sleep`**, jolla voidaan luoda halutun pituisia viiveitä. Tämä funktio on paljon käytössä grafiikan ja animaation yhteydessä. Vaikka säie luotaisiin runnable-rajapintaa käyttäen, Thread-luokan funktiot ovat käytettävissä.

start()	Aloittaa säikeen suorituksen
run()	Säikeen ohjelmakoodi tähän funktioon
stop()	Keskeyttää säikeen suorittamisen
resume()	Jatkaa säikeen suorittamista
suspend()	Lakkauttaa säikeen
sleep()	"Nukuttaa" säikeen (viive)
setPriority()	Asettaa säikeen prioriteetin

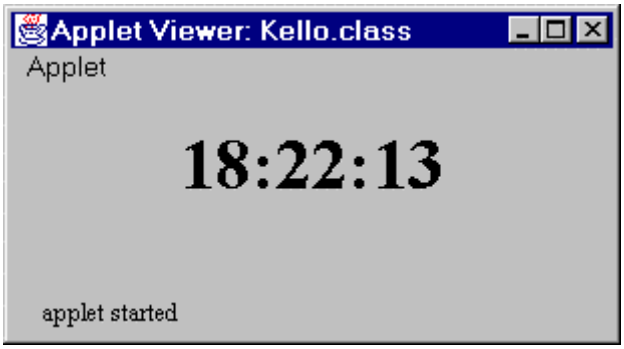
Sleep-funktion käyttö vaatii try-catch-lauseen käyttöä. Funktio generoi epäonnistuessaan poikkeutustilanteen InterruptedException.

```
try
{
 Thread.sleep(1000);
 repaint();
} catch (InterruptedException e) {}
```

Prioriteetin avulla voidaan määrätä minkä arvoisena säie näkyy käyttöjärjestelmälle. Mitä pienempi prioriteetti, sitä harvemmin säie saa prosessoriaikaa. Funktiolla **`setPriority`** on kolme mahdollista arvoa, jolla se voi asettaa prioriteetin.

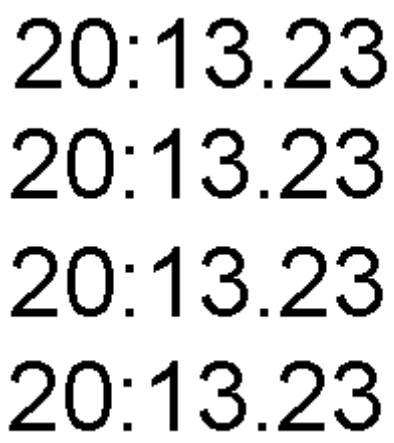
MAX_PRIORITY	Suurin prioriteetti
MIN_PRIORITY	Pienin prioritetti
NORM_PRIORITY	Normaali prioriteetti

Säikeiden avulla on helppo toteuttaa esimerkiksi digitaalinen kello, koska sleep-funktiolla voidaan nukkua sekunnin ajan tulostuksen välissä. Digitaalinen kello on lisäksi hyvä esimerkki, miksi säikeitä pitää Javassa käyttää. Ilman niitä kellon tekeminen ei edes onnistu. Kello-ohjelman lopullinen ja toimiva versio on esitetty luvussa 13. Seuraavassa on kuvattu ohjelman pääkohdat säikeen kannalta.




```
// Start-funktiot aloittaa säikeen
public void start()
 {
  if (kello == null) {
  Thread kello = new Thread(this);
  kello.start();
 }
}

//   Run-funktiossa   odotetaan   sekunti   ja   pyydetään
päivittämään kuvaruutu
public void run()
{
 while (true)
 try {
  Thread.sleep(1000);
  repaint();
 } catch (InterruptedException e) {}
}

// Paint-funktio tulostaa kellonajan
 public void paint(Graphics g)
 {
   Calendar cal = Calendar.getInstance();
   g.setFont(new Font("Arial",Font.PLAIN,48));
  g.drawString(DateFormat.getTimeInstance().format(new
    Date()), 10, 60);

 }
}
```

Edellisessä esimerkissä näytölle tulostuu vain yksi kello. Säikeitä voi olla aktivoituna useampia, joten kellojakin voi olla useampia. Seuraavassa esimerkissä on käytetty hyväksi edellisen esimerkin tekniikkaa. Kellojen tuottamisesta huolehtii tällä kertaa luokka nimeltä Kellothread. Kuvassa ohjelman tulos:




```
// init-funktiossa asetetaan gridlayout ja asetetaan
paneeliin 4 paneelia (KelloThread)
public void init()
 {
  setLayout(new GridLayout(4,1));
  add (new KelloThread());
  add (new KelloThread());
  add (new KelloThread());
  add (new KelloThread());
 }
}


// Säie, joka vastaa kellonajan tulostuksesta.
Muodostimessa asetetaan säie

class KelloThread extends Panel implements Runnable
{
 public KelloThread()
 {
  Thread kello = new Thread(this);
  kello.start();
 }

// Run-funktiossa odotetaan sekunti ja pyydetään
// päivittämään kuvaruutu
```

```
 public void run()
 {
  while (true)
  try {
   Thread.sleep(1000);
   repaint();
  } catch (InterruptedException e) {}
 }

// Paint-funktio tulostaa kellonajan
 public void paint(Graphics g)
 {
  Calendar cal = Calendar.getInstance();
  g.setFont(new Font("Arial",Font.PLAIN,48));
  g.drawString(DateFormat.getTimeInstance().format(new
    Date()), 10, 60);
}
}
```

10.6 Synkroniset säikeet

Kun kaksi säiettä käyttää samaan aikaan samaa ohjelmakoodia, saattaa syntyä odottamattomia ongelmia. Jos ohjelmakoodi on sama, yleensä myös muuttujat ovat samoja. Esimerkiksi, jos tarkoituksena on, että vain yksi säie saa asettaa muuttujan arvon ja suorittaa tärkeän ohjelmakoodin, muiden säikeiden pitää ohittaa ohjelmakoodi muuttujan arvon perusteella.

```
boolean lippu;

if (lippu != true)
{
 // suorita tärkeä koodi
 lippu = false;
}
```

Koodi näyttää ihan hyvältä, kun sitä vilkaisee nopeasti. Mutta kun muistaa, että säikeitä suoritetaan rinnakkain, ei voi olla varma missä kohdassa ohjelmakoodia muut säikeet ovat silloin, kun asetetaan muuttuja lippu arvoon tosi. Varmemmaksi ohjelmakoodia voisi tehdä siten, että sijoitus `lippu = false` suoritetaan heti ensimmäisenä. If-lause kuitenkin koostuu konekielitasolla useammasta käskystä eli jo tässäkin välissä joku ylimääräinenkin säie pääsee if-lauseen sisälle.

```
boolean lippu;
```

```
 if (lippu != true) {
  lippu = false;
  // suorita tärkeä koodi
 }
```

Javassa on kuitenkin täysin varma tekniikka, jolla voidaan varmistaa, että vain yksi säie kerrallaan suorittaa tiettyä koodia (tästä tekniikasta on ollut puhetta jo luvussa 3). Javan kielioppi tuntee varatun sanan **`synchronized`**.

Synchronized-sana tahdistaa funktion tai lohkon siten, että vain yksi säie pystyy suorittamaan sitä kerrallaan. Edellä olleen esimerkin voisi siten suojata kahdella eri tavalla.

Funktiolla:

```
boolean lippu;

public synchronized void koodi()
{
 if (lippu != true) {
  lippu = false;
  // suorita tärkeä koodi }
}
```

Koodilohkolla:

```
boolean lippu;
synchronized (olion tai luokan nimi)
{
 if (lippu != true) {
  lippu = false;
 // suorita tärkeä koodi }
}
```

Myös yksittäinen muuttuja voidaan tahdistaa. Tosin se ei tapahtu synchronized-sanalla, vaan varatulla sanalla **`threadsafe`**. Muuttujan esittelyn yhteydessä tämä varattu sana kirjoitetaan ennen tietotyypin esittelyä.

```
threadsafe boolean lippu;
```

Jotakin näistä tavoista kannattaa käyttää, kun tahdistettuja säikeitä tarvitaan. Tapauksesta riippuu, mikä on paras vaihtoehto.

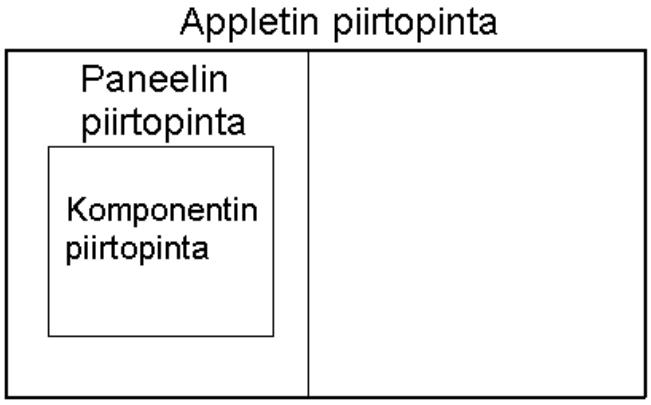
11. Grafiikka

Grafiikkaominaisuudet ovat Javassa kehittyneitä. Ja mikäpä sen parempaa, koska grafiikka on ehdottomasti Java-kielisten sovellusten tärkeimpiä ominaisuuksia WWW-maailmassa. Valmiit grafiikkafunktiot mahdollistavat helpon tavan grafiikan tuottamiseen, esimerkiksi viivojen ja kaarien sekä täytettyjen kuvioiden piirtämiseen.

11.1 Piirtopinta

Piirtopinta (*drawsurface*) on pinta, johon voidaan piirtää graafiikkaluokan funktioilla. Sen ulkopuolelle ohjelmat eivät voi koskaan piirtää. Piirtopintana Javassa toimii yleensä paneeli tai ikkuna. Joissakin tapauksissa myös Canvas-elementti voidaan laskea piirtopinnaksi.

Appletin piirtopinnan koko asettuu <APPLET>-tagin yhteydessä tai myöhemmin ohjelmassa `resize`-funktiolla. Komponenttien lisääminen containeriin ei vaikuta appletin piirtopinnan kokooon. Mikäli appletin containeriin lisätään uusi paneeli, sen piirtopinnan koko on riippuvainen appletin piirtopinnan koosta sekä komponenttien lukumäärästä ja layoutista. Komponenttien piirtopinnan koko on riippuvainen paneelin koosta sekä komponenttien lukumäärästä ja layoutista. Kuvasssa appletin piirtopinta on esitetty paksulla mustalla viivalla. Paneeleita kuvassa on kaksi, ja kummankin koko on puolet appletin piirtopinnasta.



11.2 Piirtopinnalle piirtäminen

Piirtopinnalle piirtäminen tapahtuu Graphics-luokan funktioiden avulla. Javan grafiikkaluokka sallii esimerkiksi suorakulmioiden piirtämisen joko täytettynä tai ääriviivoilla.

Piirtopinnalla operoidaan aina **koordinaateilla**. Koordinaatit ovat suhteellisia, joten vasen yläreuna on 0,0 (x,y). Oikean alareunan koordinaatit määräytyvät piirtoalueen koon mukaan, esimerkiksi 200,200. Katso kuva.

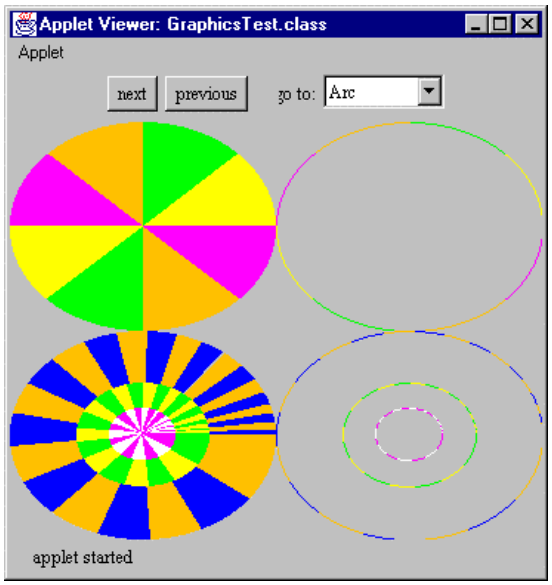
Paint-funktion tarkoituksena on toteuttaa automaattinen piirtopinnan päivittäminen. Se on yksi tapa piirtopinnalle piirtämiseen. Paint-funktio on takaisinkutsuttavafunktio, joten se täytyy ylikirjoittaa. Paint-funktio palauttaa Graphics-luokan olion. Olion funktioilla piirretään piirtopinnalle grafiikkaa, esimerkiksi suorakulmio.

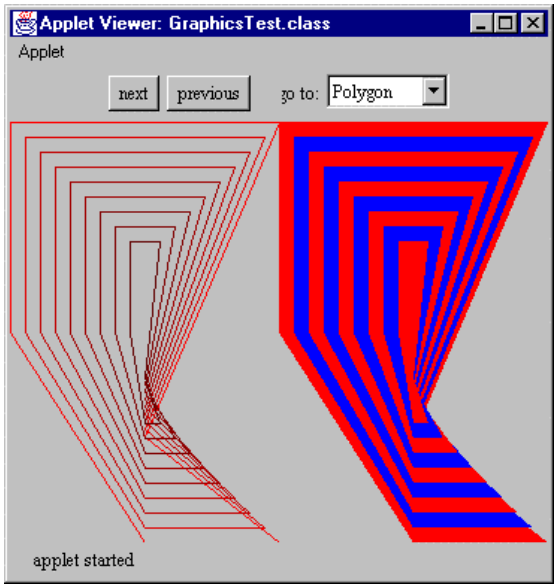
```
public void paint (Graphics g)
{
  g.drawRect(0, 0, 100, 100);
}
```

Component-luokassa on paint-funktion lisäksi toinenkin funktio, joka mahdollistaa piirtopinnalle piirtämisen. Se on nimeltään **`getGraphics`**. Nimensä mukaisesti funktio hakee Graphics-olion, jonka avulla piirtäminen sitten tapahtuu. Edellä tapahtunut suorakulmion piirtäminen drawRect-funktiolla voitaisiin toteuttaa jo init-funktion aikana käyttäen apuna getGraphics-funktiota.

```
public void init()
{
 Graphics g = getGraphics();
 g.drawRect(0, 0, 100, 100);
}
```

Javassa on kahdentyyppisiä piirtämiseen tarkoitettuja funktioita. Toinen ryhmä muodostuu täyttöfunktioista ja toinen piirtofunktioista. Täyttöfunktioissa kuviot täytetään aina piirtovärillä, piirtofunktioissa piirretään vain kuvion ääriviivat. Java-ohjelmointipaketissa on sovellus nimeltä **graphicsTest**, joka esittelee näyttävästi grafiikkaominaisuuksia (katso kuva).





11.3 Värien asettaminen

Piirtoväri asetetaan Graphics-olion funktiolla **`setColor`**. Parametriksi sille annetaan Color-luokan muuttuja, joka kertoo värin. Kun uusi piirtoväri annetaan, niin sitä käytetään kaikissa piirtopinnalle kohdistuvissa operaatioissa. Väri on käytössä kunnes asetetaan uusi. Esimerkiksi punaisen piirtovärin asettaminen tapahtuu seuraavasti:

```
setColor(Color.red);
```

Taustavärin asetukseen ei ole Graphics-luokassa funktiota, vaan pitää käyttää Component-luokan funktiota **`setBackground`**. Taustaväri määrätään myös Color-luokan muuttujalla. Taustavärin voi muuttaa tietenkin jo init()-funktiossa.

```
setBackground(Color.blue);
```

Nykyisen piirtovärin ja taustavärin saa tietoonsa funktioilla **`getColor`** ja **`getBackground`**. Funktiot palauttavat Color-olion.

```
Color piirtovari = getColor();
Color taustavari = getBackground();
```

Esimerkiksi piirtovärin asettaminen jokaiselle kuviolle voidaan toteuttaa paint-funktiossa.

```
public void paint (Graphics g)
{
 setColor(Color.blue);  // piirtoväriksi sininen
 g.drawRect(0, 0, 10, 10);
 setColor(Color.red);   // piirtoväriksi punainen
 g.drawRect(10, 10, 20, 20);
}
```

Edellisissä tapauksissa värin asettamiseen käytettiin Color-oliossa sijaitsevia jäsenmuuttujia. Näitä olivat esimerkiksi red (punainen) ja blue (sininen). Color-olio sisältää muuttujissaan useita valmiita värejä, joista voi valita sopivan.

white	valkoinen
lightGray	vaalean harmaa
gray	harmaa
darkGray	tumman harmaa
black	musta
red	punainen
pink	vaalean punainen
orange	oranssi
yellow	keltainen
green	vihreä
blue	sininen

Jos haluttua väriä ei löydy olion muuttujista, voi R-, G- ja B-komponentti yhdistelmällä rakentaa haluamansa värin. Nämä kolme komponenttiä mahdollistavat jokaisen värin, jota videokortti tukee (esimerkiksi yli 16 miljoonaa väriä). Valkoinen väri saadaan, kun kaikki komponentit ovat maksimiarvoissaan ja musta väri saadaan, kun kaikki komponentit ovat nollia.

- R on punainen värikomponentti
- G on vihreä värikomponentti
- B on sininen värikomponentti

Color-olion muodostaminen tapahtuu tietenkin new-operaattoria käyttäen. Oliota ei tarvitse muodostaa, jos käytetään valmiita jäsenmuuttujia, mutta mikäli halutaan muodostaa oma väri RGB-väriyhdistelmällä, tulee olio määritellä. Olion määrittelyyn on olemassa kolme eri tapaa:

Color(int, int, int) välillä 0-255	R, G, ja B ovat värikomponentit
Color(int)	RGB on väriyhdistelmä komponenteista seuraavasti: R = bitit 16-23 G = bitit 8-15 B = bitit 0-7
Color(float, float, float) 0.0-1.0	R, G, ja B ovat värikomponentit välillä

Yleisin muodostimista on ensimmäinen. Esimerkiksi väri musta saadaan, kun parametrit ovat nollia.

```
Color musta = Color(0, 0, 0);
```

Väriä käytetään seuraavasti:

```
public void paint (Graphics g)
{
 Color musta = Color(0, 0, 0);
 setColor(musta);
 g.drawRect(10, 10, 20, 20);
}
```

11.4 Piirtofunktiot

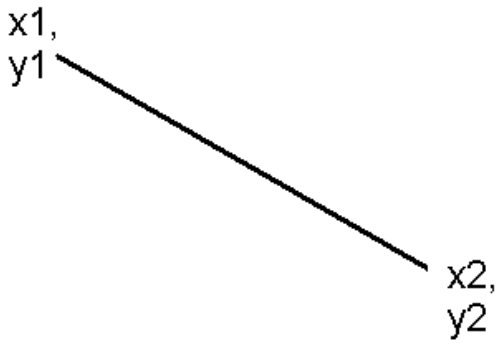
Java mahdollistaa seuraavien ei-täytettävien kuvioiden piirtämisen. Piirtäminen piirtopinnalle tapahtuu Graphics-olion avulla esimerkiksi paint-funktiossa.

- viiva
- suorakulmio
- pyöreäkulmainen suorakulmio
- 3-ulotteinen suorakulmio

- kaari
- ympyrä
- ellipsi
- monikulmio

Viivojen piirtämiseen on tarkoitettu funktio nimeltä **`drawLine`**. Viiva piirretään aloituskoordinaateista (x1,y1) lopetuskoordinaatteihin (x2,y2). Funktion parametrit ovat:

int x1	X-aloituskoordinaatti
int y2	Y-aloituskoordinaatti
int x2	X-lopetuskoordinaatti
int y2	Y-lopetuskoordinaatti




```
g.drawLine(10, 10, 100, 100);
```

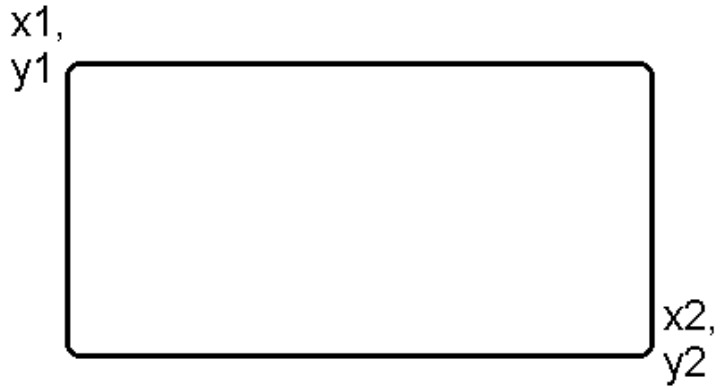
Suorakulmioiden piirtämiseen on tarkoitettu funktio nimeltä **`drawRect`**. Suorakulmio piirretään alkaen aloituskoordinaateista x1,y1. Suorakulmion koon määrää leveys ja korkeus. Suorakulmiosta piirretään piirtovärillä ääriviivat, mutta sisältö täytetään taustavärillä. Funktion parametrit ovat:

int x1	X-aloituskoordinaatti
int y1	Y-aloituskoordinaatti
int W	Suorakulmion leveys
int H	Suorakulmion korkeus

```
g.drawRect(100, 100, 100, 100);
```

Pyöreäkulmaisten suorakulmioiden piirtämiseen on tarkoitettu funktio nimeltä **`drawRoundRect`**. Suorakulmio piirretään alkaen aloituskoordinaateista x1,y1. Suorakulmion koon määrää leveys ja korkeus. Reunan pyöristykseen vaikuttavat parametrit arcW ja arcH, jotka kertovat, kuinka suuri (ympyrän)kaari reunaviivojen väliin piirretään. Suorakulmiosta piirretään piirtovärillä ääriviivat, mutta sisältö täytetään taustavärillä. Funktion parametrit ovat:

int x1	X-aloituskoordinaatti
int y1	Y-aloituskoordinaatti
int W	Suorakulmion leveys
int H	Suorakulmion korkeus
int arcW	Kaaren leveys
int arcH	Kaaren korkeus




```
g.drawRoundRect(100, 100, 100, 100, 20, 20);
```

3-ulotteisten suorakulmioiden piirtämiseen on tarkoitettu funktio nimeltä **`draw3DRect`**. 3D-suorakulmio piirretään alkaen aloituskoordinaateista x1,y1. Suorakulmion koon määrää leveys ja

korkeus. Suorakulmio voidaan kohottaa (tosi) tai upottaa (epätosi). 3D- suorakulmion avulla voi tehdä esimerkiksi keinotekoisia painonappeja. Funktion parametrit ovat:

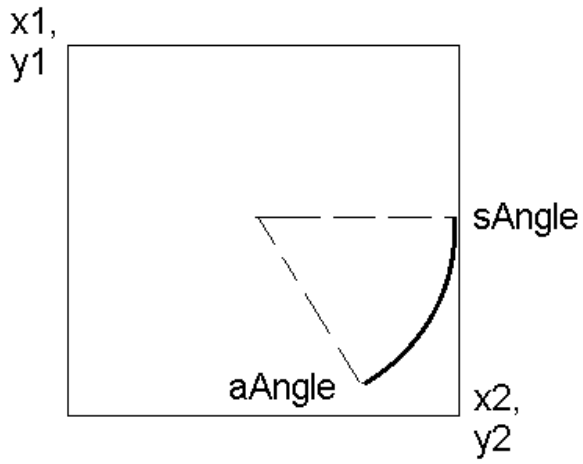
int x1	X-aloituskoordinaatti
int y1	Y-aloituskoordinaatti
int W	Suorakulmion leveys
int H	Suorakulmion korkeus
boolean raised	Kohotus/upotus

TOSI EPÄTOSI

```
g.draw3DRect(100, 100, 100, 100, true);
```

Kaarien piirtämiseen on tarkoitettu funktio nimeltä **`drawArc`**. Kaari piirretään näennäisen suorakulmion avulla, joka rajoittaa kaaren. Kaaren pituus määräytyy parametreilla sAngle (aloituskulma) ja aAngle (kaaren kulma). Kaaren nollakulma on kello kolmen kohdalla, ja positiivinen suunta on myötäpäivään. Kaaresta piirretään piirtovärillä ääriviivat. Funktion parametrit ovat:

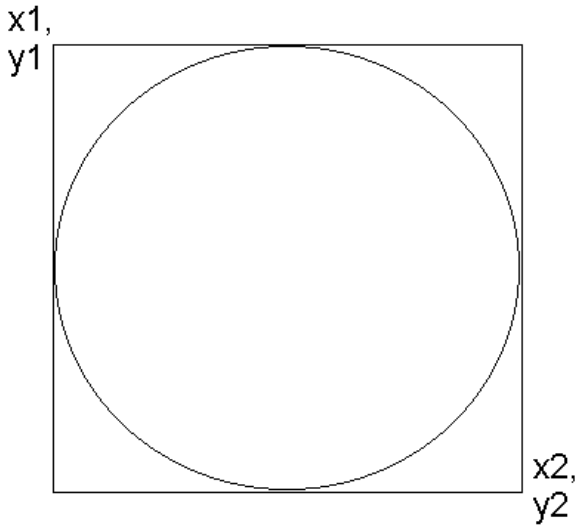
int x	Näennäisen suorakulmion X-koordinaatti
int y	Näennäisen suorakulmion Y-koordinaatti
int W	Näennäisen suorakulmion leveys
int H	Näennäisen suorakulmion korkeus
int sAngle	Kaaren aloituskohta
int aAngle	Kaaren kulma (suhteellinen)




```
g.drawArc(100, 100, 100, 100, 0, 100);
```

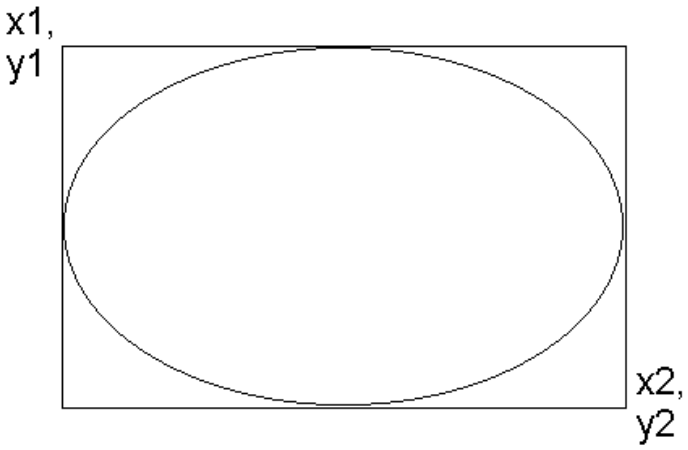
Ympyrän piirtämiseen on tarkoitettu funktio nimeltä **`drawOval`**. Ympyrä piirretään näennäisen suorakulmion avulla. Näennäisen suorakulmion tulee olla neliö, eli korkeuden ja leveyden tulee olla samat. Ympyrästä piirretään piirtovärillä ääriviiva, mutta sisältö täytetään taustavärillä. Funktion parametrit ovat:

int x	Näennäisen suorakulmion X-koordinaatti
int y	Näennäisen suorakulmion Y-koordinaatti
int W	Näennäisen suorakulmion leveys
int H	Näennäisen suorakulmion korkeus




```
g.drawOval(100,100,50,50);
```

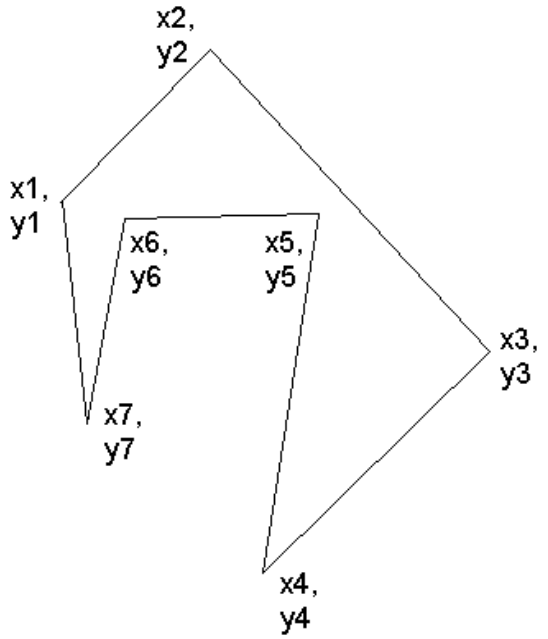
Ellipsin piirretään on myös funktiolla **`drawOval`**. Näennäinen suorakulmio ei saa olla neliö, eli korkeuden ja leveyden tulee olla erisuuret.




```
g.drawOval(100,100,100,50);
```

Monikulmion piirtämiseen on tarkoitettu funktio nimeltä **`drawPolygon`**. Monikulmio tehdään siten, että määritetään useita xy-koordinaatteja (pisteitä), joiden välille piirretään viivat. X- ja Y-koordinaatit annetaan erillisissä taulukoissa, ja koordinaattiparien eli pisteiden määrän määrää parametri nPoints. Funktion parametrit ovat:

int xPoints[]	X-koordinaatit
int yPoints[]	Y-koordinaatit
int nPoints	Pisteiden määrä




```
int x[2];
int y[2];
g.drawPolygon(x, y, 2);
```

Monikulmion piirtämiseen on myös tarkoitettu funktio nimeltä **`drawPolygon`**, jonka ainoana parametrina on Polygon-olio. Monikulmio tehdään siten, että alustetaan uusi Polygon-olio, johon määritetään useita xy-koordinaattipareja eli pisteitä. Pisteiden välille piirretään viivat samoin kuin edellä esitetyllä drawPolygon-funktiolla.

Polygon-olio luodaan muodostimella Polygon(). Olio ei sisällä valmiiksi yhtäkään xy-koordinaattiparia, joten tarvitaan avuksi funktio **`addPoint`**. Polygon-olio voidaan luoda myös siten, että muodostinta kutsuttaessa kerrotaan xy-koordinaatit. Muodostimen parametrit ovat samat kuin edellä olevan drawPolygon-funktion.

```
Polygon p = new Polygon();
p.addPoint(100, 100) ;  // lisätään pisteet
drawPolygon(p);
```

tai

```
Polygon p = new Polygon(x[], y[], 2);
drawPolygon(p);
```

11.5 Täyttöfunktiot

Java mahdollistaa seuraavien täytettävien kuvioiden piirtämisen. Piirtäminen piirtopinnalle tapahtuu Graphics-olion avulla esimerkiksi paint-funktiossa.

- suorakulmio
- pyöreäkulmainen suorakulmio
- 3-ulotteinen suorakulmio
- kaari
- ympyrä
- ellipsi
- monikulmio

Suorakulmioiden täyttämiseen on tarkoitettu funktio nimeltä **`fillRect`**. Suorakulmio piirretään alkaen aloituskoordinaateista x1,y1. Suorakulmion koon määrää leveys ja korkeus. Suorakulmion sisältö täytetään piirtovärillä. Funktion parametrit ovat:

int x1	X-aloituskoordinaatti
int y1	Y-aloituskoordinaatti
int W	Suorakulmion leveys
int H	Suorakulmion korkeus

```
g.fillRect(100, 100, 100,100);
```

Pyöreäreunaisten suorakulmioiden täyttämiseen on tarkoitettu funktio nimeltä **`fillRoundRect`**. Suorakulmio piirretään alkaen aloituskoordinaateista x1,y1. Suorakulmion koon määrää leveys ja korkeus. Reunan pyöristykseen vaikuttavat parametrit arcW ja arcH, jotka

kertovat, kuinka suuri (ympyrän)kaari reunaviivojen väliin piirretään. Suorakulmion sisältö täytetään piirtovärillä. Funktion parametrit ovat:

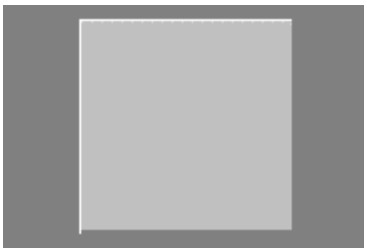
int x1	X-aloituskoordinaatti
int y1	Y-aloituskoordinaatti
int W	Suorakulmion leveys
int H	Suorakulmion korkeus
int arcW	Kaaren leveys
int arcH	Kaaren korkeus

```
g.fillRoundRect(100, 100, 100, 100, 20, 20);
```

3-ulotteisen suorakulmion täyttämiseen on tarkoitettu funktio nimeltä **`fill3DRect`**. 3D-suorakulmio piirretään alkaen aloituskoordinaateista x1,y1. Suorakulmion koon määrää leveys ja korkeus. Suorakulmio voidaan kohottaa (tosi) tai upottaa (epätosi). Suorakulmion sisältö täytetään piirtovärillä. Funktion parametrit ovat:

int x1	X-aloituskoordinaatti
int y1	Y-aloituskoordinaatti
int W	Suorakulmion leveys
int H	Suorakulmion korkeus
boolean	Kohotus/upotus

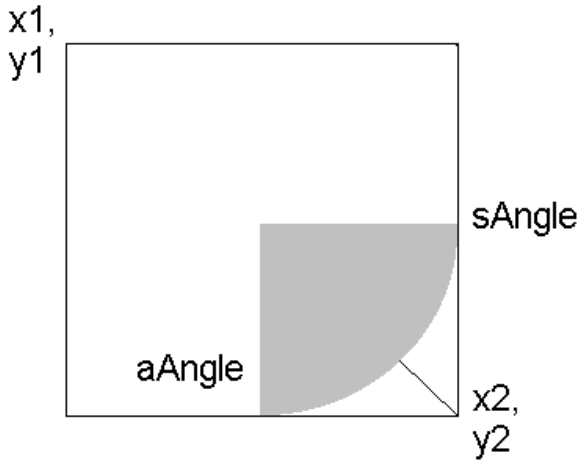
TOSI EPÄTOSI



```
g.fill3DRect(100, 100, 100, 100, true);
```

Kaarien piirtämiseen ja täyttämiseen on tarkoitettu funktio nimeltä **`fillArc`**. Kaari piirretään näennäisen suorakulmion avulla, joka rajoittaa kaaren. Kaaren pituus määräytyy parametreilla sAngle (aloituskulma) ja aAngle (kaaren kulma). Kaaren nollakulma on kello kolmen kohdalla, ja positiivinen suunta on myötäpäivään. Kaaren sisältö täytetään piirtovärillä. Parametrit ovat:

int x	Näennäisen suorakulmion X-koordinaatti
int y	Näennäisen suorakulmion Y-koordinaatti
int W	Näennäisen suorakulmion leveys
int H	Näennäisen suorakulmion korkeus
int sAngle	Kaaren aloituskohta
int aAngle	Kaaren kulma (suhteellinen)

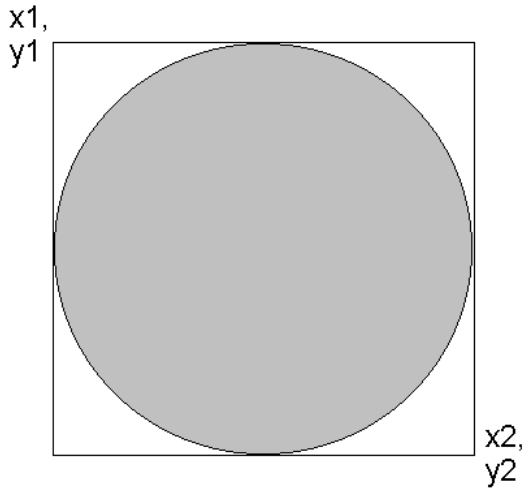



```
g.fillArc(100, 100, 100, 100, 0, 100);
```

Ympyrän piirtämiseen ja täyttämiseen on tarkoitettu funktio nimeltä **`fillOval`**. Ympyrä piirretään näennäisen suorakulmion avulla. Näennäisen suorakulmion tulee olla neliö, eli

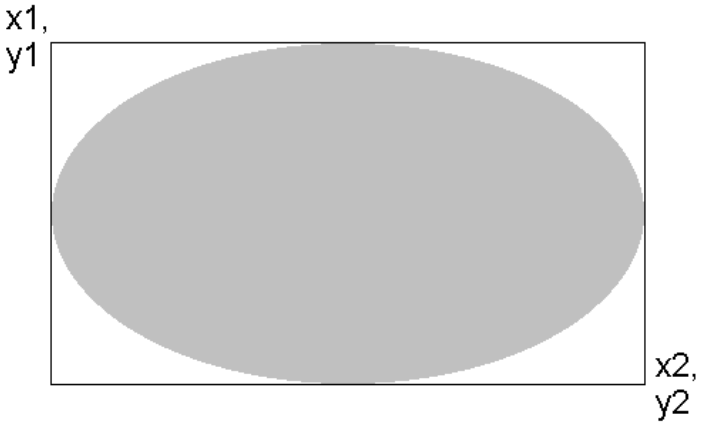
korkeuden ja leveyden tulee olla samat. Ympyrän sisältö täytetään piirtovärillä. Funktion parametrit ovat:

int x	Näennäisen suorakulmion X-koordinaatti
int y	Näennäisen suorakulmion X-koordinaatti
int W	Näennäisen suorakulmion leveys
int H	Näennäisen suorakulmion korkeus




```
g.fillOval(100,100, 50,50);
```

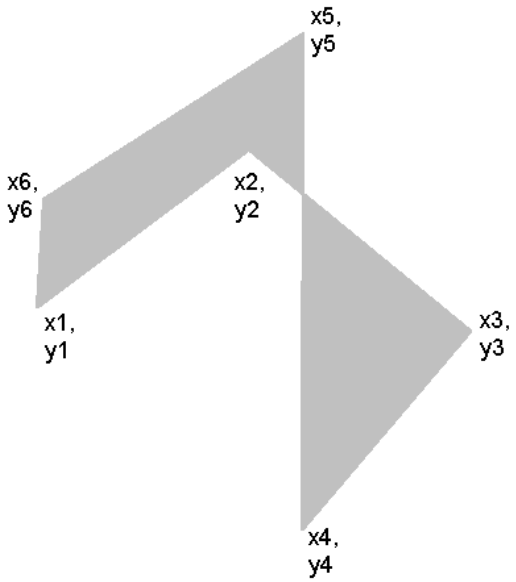
Ellipsin piirtämiseen ja täyttämiseen on tarkoitettu myös **`fillOval`**-funktio. Näennäinen suorakulmio ei saa olla neliö, eli korkeuden ja leveyden tulee olla erisuuret.




```
g.fillOval(100, 100, 100, 50);
```

Monikulmion piirtämiseen ja täyttämiseen on tarkoitettu funktio nimeltä **`fillPolygon`**. Monikulmio tehdään siten, että määritetään useita xy-koordinaatteja (pisteitä), joiden välille piirretään viivat. X- ja Y-koordinaatit annetaan erillisissä taulukoissa, ja koordinaattiparien eli pisteiden määrän määrää parametri nPoints. Monikulmion sisältö täytetään piirtovärillä. Funktion parametrit ovat:

int xPoints[]	X-koordinaatit
int yPoints[]	Y-koordinaatit
int nPoints	Pisteiden määrä




```
int x[2];
int y[2];
g.drawPolygon(x, y, 2);
```

Monikulmion piirtämiseen ja täyttämiseen on myös tarkoitettu funktio nimeltä **`fillPolygon`**, jonka ainoana parametrina on Polygon-olio. Monikulmio tehdään siten, että alustetaan uusi Polygon-olio, johon määritetään useita xy-koordinaatteja (pisteitä). Pisteiden välille piirretään viivat samoin kuin edellä esitetyllä fillPolygon-funktiolla.

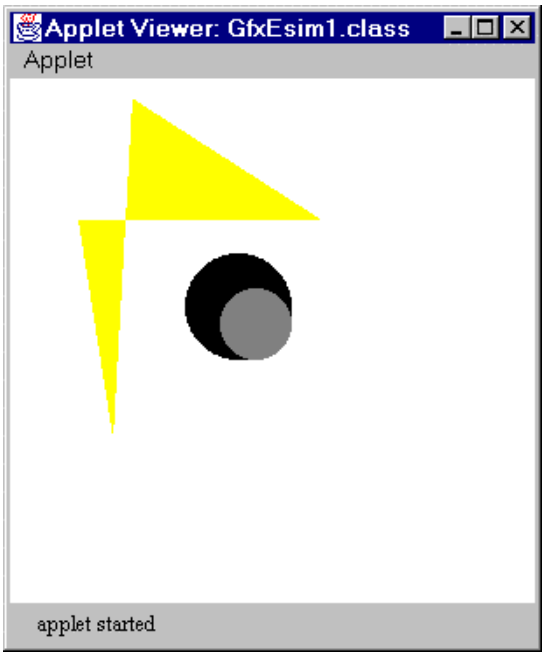
Polygon-olio luodaan muodostimella Polygon(). Olio ei sisällä yhtäkään xy-koordinaattia, joten tarvitaan avuksi funktiota **`addPoint`**. Polygon-olio voidaan luoda myös siten, että muodostinta kutsuttaessa kerrotaan xy-koordinaatit. Muodostimen parametrit ovat samat kuin edellä olevan fillPolygon-funktion.

```
Polygon p = new Polygon();
p.addPoint(100, 100) ;  // lisätään pisteet
fillPolygon(p);
```

tai

```
Polygon p = new Polygon(x[], y[], 2);
fillPolygon(p);
```

Java-ohjelma **GfxEsim1.java** esittelee täyttö- ja piirtofunktioiden toimintaa. Ohjelmassa piirretään paint-funktiossa monikulmio, johon pisteet antaa käyttäjä hiirellä. Samassa funktiossa piirretään myös kaksi ympyrää, toinen mustalla ja toinen harmaalla. Katso kuva.




```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class GfxEsim1 extends Applet
{
 Polygon p;

 public void init()
 {
  p = new Polygon();
  setBackground(Color.white);
  resize(300,300);
  enableEvents(AWTEvent.MOUSE_EVENT_MASK);
 }
```

```
 public void paint(Graphics g)
 {
  g.setColor(Color.yellow);  // Väriksi keltainen
  g.fillPolygon(p);          // Piirrä monikulmio
  g.setColor(Color.black);   // Väriksi musta
  g.fillOval(100,100,60,60); // Piirrä ympyrä
  g.setColor(Color.gray);    // Väriksi harmaa
  g.fillOval(120,120,40,40); // Piirrä ympyrä
 }


 public void processMouseEvent(MouseEvent e)
 {
  // Uuden viivan koordinaatit ovat x ja y
  if (e.getID() ==  e.MOUSE_CLICKED)
  {
   p.addPoint(e.getX(), e.getY());   // Lisää polygon
                                     olioon
   repaint();                     // Päivitä ikkuna
  }
  super.processMouseEvent(e);

 }
}
```

11.6 Alueiden käsittely

Suorakulmaisten alueiden käsittely piirtopinnalla on eräs tärkeimpiä toimenpiteitä. Alueiden käsittelyyn kuuluu alueiden kopiointi ja puhdistus (nollaus).

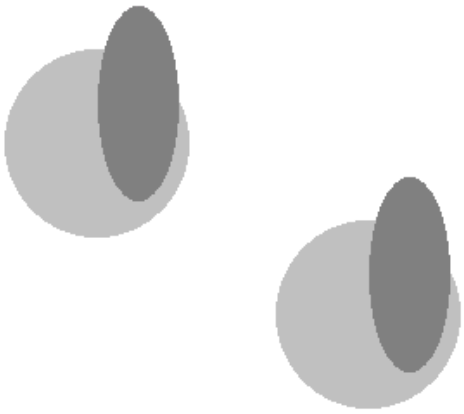
Alueen nollaus on samaa kuin taustavärillä täyttäminen. Tähän tehtävään on tarkoitettu funktio nimeltä **`clearRect`**. Funktio toimii suorakulmaisesti, joten sille annetaan parametreiksi aloituskoordinaatit, leveys ja korkeus. Funktio on hyödyllinen esimerkiksi animaation teossa, koska sillä voidaan helposti pyyhkiä kuva piirtopinnalta. Funktion parametrit ovat

int X1	X-aloituskoordinaatti
int Y1	Y-aloituskoordinaatti
int W	Suorakulmion leveys
int H	Suorakulmion korkeus

```
g.clearRect(100, 100, 50, 50);
```

Alueiden kopiointiin on suunniteltu funktio nimeltä **`copyArea`**. Funktio kopioi piirtopinnalla suorakulmaisen alueen paikasta A paikkaan B. *Kopiointi tapahtuu vain piirtopinnasta piirtopintaan!* Muistista ei voi kopioida alueita piirtopinnalle. Funktiota voi hyödyntää esimerkiksi animaatiossa ja vierityksessä. Funktion parametrit ovat

int Lx	Lähteen X-koordinaatti
int Ly	Lähteen Y-koordinaatti
int W	Suorakulmion leveys
int H	Suorakulmion korkeus
int Kx	Kohteen X-koordinaatti
int Ky	Kohteen Y-koordinaatti



```
g.copyArea(10,10, 50, 50, 200, 200);
```

Edellä esitetyt funktiot toimivat yhteistyössä hyvin. Koska funktiot kohdistavat toimintonsa tunnettuun suorakulmaiseen alueeseen, kuvaruudun päivittäminen on järkevämpää suorittaa **`update`**-funktiolla kuin paint-funktiolla. Update-funktiosta kerrotaan lisää luvussa *12 Multimediaa Javalla*.

11.7 Tekstin tulostaminen

Graphics-luokan tekstintulostusfunktiot ovat yksinkertaisia, eivätkä tunne minkäänlaisia muotoilumerkkejä (ei edes rivinvaihtoa), joilla tekstintulostusta voitaisiin ohjata. Tämä pitää tietenkin huomioida tulostettaessa siten, että on tiedettävä mistä kohtaa merkkijono on katkaistava, että se mahtuu piirtopinnalle kokonaisuudessaan.

Tekstin tulostaminen tapahtuu aina nykyisellä fontilla ja piirtovärillä. Piirtoväri asetetaan setColor-funktiolla siten kuin edellä on esitetty. Appletin käynnistyessä oletusfonttina on New Times Roman kooltaan 10. Fontti ei vaihdu, vaikka käytettäisiin HTML-kielen Hx-tagia!

Tekstin perustulostusfunktio on nimeltään **`drawString`**, jolla tulostetaan merkkijono annetusta koordinaateista alkaen. Merkkijono tulostetaan kokonaisuudessaan samalle riville, muotoilumerkkejä ei huomioida. Funktion parametrit ovat:

String s	Merkkijono
int x	X -koordinaatti
int y	Y -koordinaatti

```
g.drawString(”Hello Web!”, 10, 10);
```

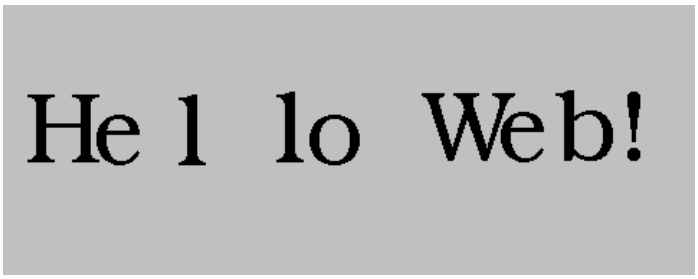
Toinen tekstintulostusfunktio on **`drawChars`**. Se tulostaa nimensä mukaisesti merkkitaulukon merkkejä. Javassa merkkijono on String-olio, mutta se saadaan muutettua merkkitaulukoksi **`toCharArray`**-funktiolla. Edellisestä funktiosta poiketen drawChars sallii merkkijonon pituuden ja aloitusmerkin asetuksen.

Char[]	Merkkitaulukko
int offset	Aloitus indeksi
int lenght	pituus (merkeissä)
int X	X -koordinaatti

int Y Y -koordinaatti

```
String teksti = ”Hello Web!”;
 g.drawChars(teksti.toCharArray(), 4, 4, 10, 10);
```

Funktio drawChars on käyttökelpoinen silloin kun halutaan tulostaa merkkejä yksitellen. Java-ohjelmointipaketin mukana tullut ohjelma nimeltään **NervousText** värisyttää annettua tekstiä. Se on toteutettu drawChars-funktiolla - jokainen merkki tulostetaan erikseen käyttäen apuna satunnaislukua.



Tekstin värin vaihto on yksi käyttökelpoinen efekti grafiikkaesityksissä. Se on näyttävä ja lisäksi helppo toteuttaa. Myös HTML tuntee värin vaihtoon liittyvän tagin <**BLINK**>, joka vilkuttaa annettua tekstiä. Vilkutus on yksinkertaisesti kahdella eri värillä piirtämistä: piirtovärillä (esimerkiksi mustalla) ja taustavärillä (esimerkiksi harmaalla). Vilkkuvaa tekstiä voidaan tulostaa esimerkiksi paint-funktiossa drawString-funktiolla. Edellä opitun mukaisesti setColor-funktio asettaa piirtovärin.

```
public void paint(Graphics g)
{
 g.setColor(Color.RED);
 g.drawString(”Tämä teksti tulostuu punaisella”,
                10, 20);
}
```

Kun tähän runkoon lisätään vielä säie, joka päivittää ikkunan piirtopintaa tietyn väliajoin ja samalla muutetaan piirtoväriä, saadaan vilkkuva teksti. Joka toisella kerralla piirtoväri on musta ja joka toisella taustaväri.

```
import java.applet.*;
import java.awt.*;

public class Blink extends Applet implements Runnable
{
 int vari;
 Thread th;

 public void init()
 {
  setFont(new Font("Arial",Font.PLAIN,24));
  vari = 0;
 }

 public void start()
 {
  th = new Thread(this);
  th.start();
 }

 public void stop()
 {
  th.stop();
  th = null;
 }

 public void run()
 {
  while (true)
```

```
  {
   repaint();
   try {
    th.sleep(500);
   } catch (InterruptedException e) { }
  }
 }

 public void paint(Graphics g)
 {
  // Vaihda piirtoväriä
  vari ^= 1;
  if (vari == 1)
   g.setColor(Color.black);
  else
   g.setColor(getBackground());

  // Tulosta teksti
  g.drawString("Tämä teksti vilkkuu", 10, 20);
 }
}
```

11.8 Tekstin tulostaminen eri fonteilla

Uuden fontin valinnassa tarvitaan avuksi Font-luokkaa. Fontti muodostetaan siten, että kutsutaan muodostinta **Font**, jossa parametrit ovat fontin nimi, tyyli ja koko. Fontin nimi voi olla esimerkiksi yksi seuraavista:

- Arial
- Monospaced
- SansSerif
- Serif

Fontin valinnan helpottamiseksi
on olemassa luokassa java.awt.Toolkit funktio **`getFontList`**, joka palauttaa merkkijonotaulukossa käyttöjärjestelmän tuntemat fontit.

Tyyli on kolmen eri tyyliyhdistelmän tulos. Kaikki tyylit (tai ei yksikään) voivat olla valittuna samaan aikaan.

- BOLD Lihavointi

- ITALIC Kursivointi
- PLAIN Normaali

Fontin koko kertoo fontin korkeuden, leveys määräytyy automaattisesti suhteessa korkeuteen. Yleisimmät pistekoot ovat:

- 10
- 12
- 14

Seuraavassa luodaan looginen Arial-fontti, joka on kursiivi ja jonka pistekoko on 14. Kun font-olio luodaan new-operaattoria käyttäen, saadaan tulokseksi aina looginen fontti. Mikäli järjestelmän fyysisistä fonteista ei löydy täysin oikeanlaista fonttia, palautetaan siitä lähinnä oleva. Joissakin tapauksissa fontti saattaa poiketa melkoisesti pyydetystä, esimerkiksi silloin, kun järjestelmässä ei ole pyydettyä fonttia.

```
Font fontti = new Font("Arial", Font.ITALIC, 14);
```

Lopuksi fontti valitaan graphics-olion nykyiseksi fontiksi funktiolla **`setFont`**.

```
g.setFont(fontti);
```

Nyt drawString- ja drawChar-funktiot tulostavat tekstit Arial-fontilla.

```
g.drawString(”Hello Web!”, 10, 10);
```

Fontin valintaa ei välttämättä tarvitse suorittaa paint-funktiossa, vaan se on mahdollista suorittaa jo init-funktiossa. Näin voidaan menetellä silloin, kun koko ohjelman suorituksen ajan käytetään samaa fonttia.

```
public void init()
{
 setFont(new Font("Arial", Font.ITALIC, 14);
}
```

Jotta tiedettäisiin kuinka monta pikseliä tulostettava merkkijono on, pitää hyödyntää FontMetrics-luokan funktiota **`stringWidth`**. FontMetrics-luokasta ei voi tehdä oliota, koska se on tyyppiä abstract. Sen vuoksi tarvitaan avuksi Component-luokan funktiota **`getFontMetrics`**.

```
String leveys = ”Tämän merkkijonon leveys lasketaan”;
Font fontti = new Font(”Arial”, Font.PLAIN, 36);
GetFontMetrics ft =getFontMetrics(fontti);
leveys = ft.stringWidth;
```

Muita hyödyllisiä FontMetrics-luokan funktioita ovat:

charWidth(int)	Palauttaa merkin leveyden
charWidth(char)	Palauttaa merkin leveyden
charsWidth(char[], int, int)	Palauttaa merkkien leveyden
getHeight()	Palauttaa fontin korkeuden
getWidths()	Palauttaa kirjainten keskileveyden
stringWidth(String)	Palauttaa merkkijonon leveyden

11.9 Tekstidemo

Vaikka tekstintulostus on yksinkertainen asia, sillä saavutetaan näyttäviä ja monipuolisia sovelluksia vebbiin. Tästä esimerkkinä toimii TekstiDemo, joka on parametriohjattu sovellus. CD:llä on valmiita HTML-tiedostoja, jotka esittelevät TekstiDemon ominaisuuksia.

```
/*

    TekstiDemo By Petri Kiuttu 1995

    Appletin parametrit:

    suorita , jossa toiminnot
     V - Vieritä vasemmalle
     O - Vieritä oikealle
     Y - Vieritä ylös
     A - Vieritä alas
     S - pysäytä
     E - esitys
     U - Seuraava sana
```

```
     F - lopeta
     C - aloita alusta

    teksti1-16  Esitystekstit
    nopeus      Esitysnopeus
*/

import java.awt.*;
import java.applet.*;
import  java.net.URL;
import java.lang.*;

public  class  TekstiDemo  extends  Applet  implements
Runnable
{

 String teksti[],suorita;
 int x=0,y=0; // Tekstin koordinaatit
 Thread th; // thread-prosessi

 // Muuttujat varattu eri toiminoille
 int count=0;
 boolean loppu=true;
 int komento;
 int time;
 int nopeus=3;
 char merkki;
 int kasky;
 Font ft;
 int suunta,sana;

 public void init()
 {
  String nimi;
  // lue parametri teksti
  teksti = new String[16];
  for (int i=0; i<16; i++) {
  nimi = "teksti"+(i+1);
  teksti[i] = getParameter(nimi); }

  // Lue parametri suorita
  suorita = getParameter("suorita");
  setFont(ft = new Font("Times New Roman",
    Font.PLAIN,48));

  try {
  if (getParameter("nopeus")!=null)
```

```
  nopeus = Integer.valueOf(getParameter("nopeus"))
   .intValue();
 } catch (NumberFormatException e) {}
}

public void start()
{

 // Aloita prosessin suorittaminen
 if (th == null) {
 th = new Thread(this);
 th.start(); }
}

public void stop()
{
 th = null;
}

public void run()
{
 // Alusta muuttujat
 count=0;
 kasky=0;
 loppu=true;

 // Aseta X- ja Y-koordinaatit
 if (suorita.charAt(0)=='V')
  { x=getBounds().width; y=getBounds().height/2; }
 if (suorita.charAt(0)=='O')
  { x=-100; y=getBounds().height/2; }
 if (suorita.charAt(0)=='Y')
  { x=10; y=getBounds().height+ft.getSize(); }
 if (suorita.charAt(0)=='A')
  { x=10; y=-ft.getSize(); }
 if (suorita.charAt(0)=='E')
  { x=10; y=getBounds().height/2;}

 // Suorita pääloopppia, kunnes F (FINISH)
 // tai C (CONTINUE) jatkaa alusta
 while (loppu)
 {
  count=0; time=0;
  if (teksti[sana] == null) sana=0;
  // Tee komennosta tokeni
  merkki = suorita.charAt(kasky);
  if (merkki == 'V') komento=3;
```

```
   if (merkki == 'O') komento=4;
   if (merkki == 'Y') komento=5;
   if (merkki == 'A') komento=6;
   if (merkki == 'S') komento=7;
   if (merkki == 'U') {sana+=1; kasky+=2; continue; }
   if (merkki == 'E') { komento=8; suunta^=1; }
   if (merkki == 'F') { loppu=false; continue; }
   if (merkki == 'C') { kasky=0; continue; }

   try {// Tee suoritusaika
    time = Integer.parseInt(suorita.substring(kasky+1,
     kasky+4),10);
   } catch (NumberFormatException n) { }

   kasky+=5;  // Osoita seuraavaa käskyä

   // Suorita toiminto
   for (int i=1; count<time; i++)
   {
    if (komento!=7)
     repaint();
    else
     count+=nopeus;
    try {
     th.sleep(100);
    } catch (InterruptedException e) { }
   }
  }
  komento=1;
}

public void paint(Graphics g)
{
 if (komento==1) {
 g.drawString(teksti[sana],x,y);
 return; }

 count+=nopeus;
 switch (komento)
 {
  case 3:      // Vieritä vasemmalle
   x-=nopeus;
   break;

  case 4:      // Vieritä oikealle
   x+=nopeus;
   break;
```

```
   case 5:      // Vieritä ylös
    y-=nopeus;
    break;

   case 6:      // Vieritä alas
    y+=nopeus;
    break;

   case 8:
    if (suunta==1)
     g.setFont(new Font("Times New Roman",
      Font.PLAIN,count));
    else
     g.setFont(new Font("Times New Roman",
      Font.PLAIN,time-count));
    break;
   }
   g.drawString(teksti[sana],x,y);
 }
}
```

TekstiDemoon on sisällytetty vieritykseen mahdollistavat parametrit. Siinä esitetty vieritys ei ole täysin "aitoa", koska jatkuva vieritys samaan suuntaan ei ole mahdollista - koordinaattiesitys tekstin sijainnista estää tämän. Tekstin vierittäminen kannattaa toteuttaa siten, että se toimii vain **yhteen suuntaan** ja on riippumaton koordinaateista. Luvussa 13 on tällainen sovellus **Scrolleri.java**.

12. Multimediaa Javalla

Luvussa opetellaan multimedian tekoa eli yhdistettyä äänen ja kuvan esittämistä. Luvun aikana perehdytään myös animaation tuottamiseen. Häiritsevän välkynnän vähentämiseen etsitään konsteja luvun loppupuolella. Luvussa esitetään myös Java-ohjelma **AnimaatioDemo.java**, jonka avulla lukijat voivat tuottaa omia animaatioesityksiä WWW-sivuilleen.

12.1 Kuvat

Javassa on suora tuki vain GIF- ja JPEG-kuville. JPEG on tiiviimpi kuva formaatti, joten sen käyttö on suositeltavaa. GIF- ja JPEG kuvien tulostaminen on helppoa, koska siihen tarvitaan vain yksi funktio. Valitettavasti muissa kuvaformaateissa olevien kuvien esittäminen on hankalaa, koska purkurutiini on ohjelmoitava itse.

Kuvien lataaminen tapahtuu applet-luokan **`getImage`**-funktiolla . Kuvan yksilöiminen suoritetaan URL-osoitteella ja kuvan nimellä. Jos kuva sijaitsee omassa kotihakemistossa, kannattaa käyttää applet-luokan funktioita getDocumentBase muodostamaan perus-URL-osoite. Kuvan nimi yksilöidään getImage-funktion toisella parametrilla. Vaikka Java tuntee vain edellä mainitut formaatit, päätteeet GIF tai JPG eivät ole itsestäänselvyys tiedoston nimessä!

```
Image kuva = getImage(getCodeBase(), "kuva1.gif");
```

Jos kuvan lataaminen tapahtuu muualla kuin appletissa, esimerkiksi Frame-oliossa, pitää käyttää Toolkit-luokan funktiota **`getImage`**. Funktion parametriksi annetaan pelkkä URL-osoite ilman erillistä tiedoston nimeä. Koska Toolkit on tyypiltään abstract, siitä ei voi tehdä oliota. Avuksi tarvitaan Component-luokan **`getToolkit`**-funktiota.

```
Image kuva = getToolkit().getImage(url);
```

Kuvien yhteydessä on vahvasti esillä luokka Image. Luokka on olemassa vaikka jotkut beta-version API:t eivät sitä tunnekaan. Image-luokassa on hyvin vähän funktioita ja ne tarvitsevat tuekseen erilaisia rajapintoja, pääasiassa **ImageObserver**-rajapinnan. Image-luokassa on kuitenkin kaksi erittäin tarpeellista funktiota: **`getWidth`** ja **`getHeight`**. Nämä funktiot palauttavat kuvan korkeuden ja leveyden. Image-luokan lisäksi on erillinen kirjasto **java.awt.image**, jossa on tarvittavat rajapinnat kuvien käsittelyyn. Kuvan käsittelystä kerrotaan hieman myöhemmin.

```
Image kuva = getImage(getCodeBase(), "kuva1.gif");
int leveys = kuva.getWidth(this);
int korkeus = kuva.getHeight(this);
```

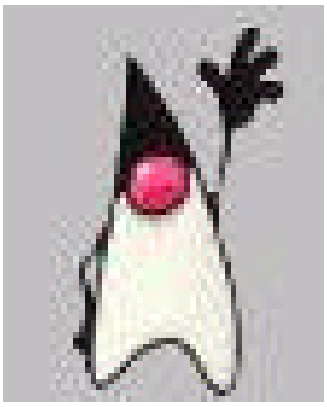
Kuva piirretään Graphics-luokan funktiolla **`drawImage(Image, int, int, Color, ImageObserver)`**. Kuva piirtyy koordinaatteihin x, y siten, että kuvan koko säilyy normaalina (ei suurenneta eikä pienennetä). ImageObserver on rajapinta, joka voidaan asettaa this-lauseella, vaikka kyseistä rajapintaa ei olisi liitetty omaan ohjelmaan. DrawImage-funktion parametrit ovat:

Image kuva	GIF-kuva
int X	X-koordinaatti
int Y	Y-koordinaatti
ImageObserver	Rajapinta

```
g.drawImage(kuva, 100, 100, this);
```

Funktiolla **`drawImage(Image, int, int, int, int, ImageObserver)`** piirtäminen tapahtuu tietylle suorakulmaiselle alueelle. Kuvan koko muuttuu, ellei suorakulmion koko ole aivan sama kuin kuvan koko. Mikäli kuva on pienempi kuin suorakulmainen alue, suurennetaan kuvaa. Jos taas suorakulmainen alue on pienempi, pienennetään kuvaa. Kuvan koon muutoksen vuoksi tämä funktio ei ole paras mahdollinen tulostusfunktio nopeisiin sovelluksiin, kuten animaatioon. Jonkinlaisen efektiin, jossa tarvitaan skaalausta, se saattaa sopia. Suurennettaessa kuvan tarkkuus kärsii melko paljon (katso seuraava kuva). Funktion parametrit ovat:

Image kuva	GIF-kuva
int X	X-koordinaatti
int Y	Y-koordinaatti
int W	Leveys
int H	Korkeus
ImageObserver	Rajapinta



```
g.drawImage(kuva, 100, 100, 200, 200, this);
```

12.2 Kuvien muokkaus

Kuvien muokkaus perustuu erilaisiin luokkiin ja rajapintoihin, jotka sijaitsevat **java.awt.image**-luokkakirjastossa. Tarkoituksena on tutustua seuraaviin asioihin.

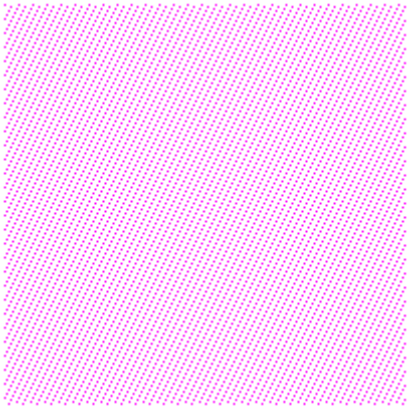
- Kuvan luonti
- Värimallit
- Suodattimet

Kuvan luonti tapahtuu Component-luokan funktiolla **`createImage`**. Funktio tarvitsee parametrikseen luokan, jossa on ImageProducer-rajapinta. Eräs tällainen on **MemoryImageSource**-luokka. MemoryImageSource-luokka käyttää taulukkoa tuottamaan kuvan pikselit. Luokassa on muodostimia kuusi, mutta tutustutaan niista vain kahteen. Kummassakin taulukko tehdään int-tietotyypillä.

MemoryImageSource(int, int, int[], int, int)
: Luo ImageProducer- olion, joka käyttää int-tyyppistä taulukkoa ja oletus RGB-värimallia (ColorModel-luokka).

MemoryImageSource(int, int, ColorModel, int[], int, int)
: Luo ImageProducer- luokan, joka käyttää int-tyyppistä taulukkoa ja RGB värimallia, joka tehdään ColorModel-luokalla.

Kuva voidaan luoda hyvin oletus RGB-värimallilla. Aluksi tehdään yksinkertainen esimerkki. Java-ohjelma **Kuva.java** täyttää 200 * 200 kokoisen alueen joka seitsemännellä pikselillä (katso kuva).



```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;

public class Kuva extends Applet
{
  int leveys, korkeus;
  int pikselit[];
  Image kuva;

  public void init()
  {
    int leveys = 200;
    int korkeus = 200;
    int pikselit[] = new int [leveys * korkeus];

    // tee pikselit
    for (int x = 0; x < leveys*korkeus; x+=7)
     pikselit[x] = (255 << 24) | (255 << 16) | 255;

    // tee kuva
    kuva = createImage(new MemoryImageSource(leveys,
                        korkeus, pikselit, 0, leveys));
  }

  public void paint(Graphics g)
  {
   g.drawImage(kuva, 1, 1, this);
  }
}
```

Värimalli (*colormodel*) kertoo minkä värinen kukin pikseli on. Java.awt.image-luokkakirjastossa on valmiina kaksi luokkaa, joihin tutustaan seuraavaksi.

- DirectColorModel
- IndexColorModel

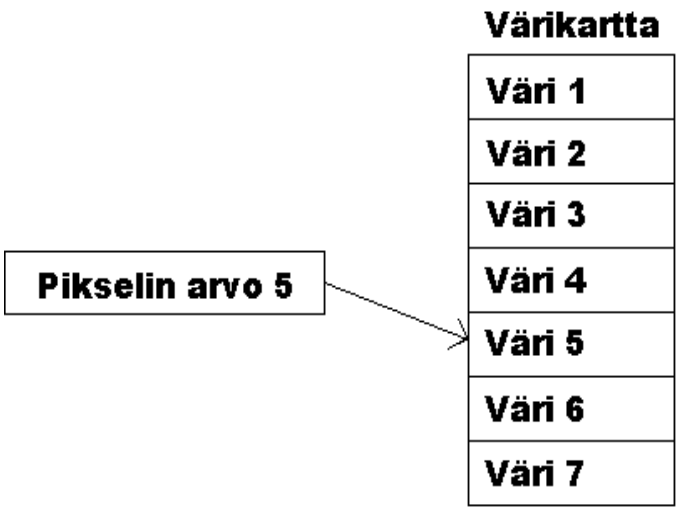
Luokka **DirectColorModel** kertoo missä suhteessa värikomponentit R, G ja B näkyvät. Se tapahtuu maskin avulla. Väriarvon ja maskin välillä tehdään AND-operaatio. Kuvassa väriarvo on vahvasti määräytynyt G ja B -komponenttien mukaan, mutta lopputulos on kuitenkin tummanpunainen. Luokalla on kaksi muodostinta, joista tässä tutustutaan muodostimeen, joka ei tarvitse alpha-parametria.

DirectColorModel(int, int, int, int)
: Muodostaa DirectColorModel-olion annetuista maskeista määrittäen näin mitkä pikselin bitit sisältävät värikomponentteja.

DirectColorModel-luokan voi liittää edelliseen esimerkkiin seuraavasti: käytetään MemoryImageSource-luokan muodostinta, joka tarvitsee parametrikseen värimallin. Pikselit annetaan väriarvoina siten, että sininen on biteissä 0-8, vihreä 8-16 ja punainen 16-24.

```
// tee pikselit
for (int x = 0; x < leveys*korkeus; x+=7)
 pikselit[x] = (255 << 24) | (255 << 16) | 255;

// Tee uusi värimalli
DirectColorModel malli = new DirectColorModel(24, 0,
                                              127, 0);

// tee kuva
kuva = createImage(new MemoryImageSource(leveys,
            korkeus, malli, pikselit, 0, leveys));
```

IndexColorModel-luokka määrittelee värikartan, jossa on erilaisia värejä. Kuvassa pikselin arvo on 5, jolloin loppullinen väri määräytyy värikartan värin 5 perusteella. Se voi olla esimerkiksi punainen, jos RGB-komponentti on 0xFF0000. Luokalla on viisi muodostinta, ja niistä keskitymme muodostimeen, jolle RGB-komponentit annetaan erillisessä taulukossa.



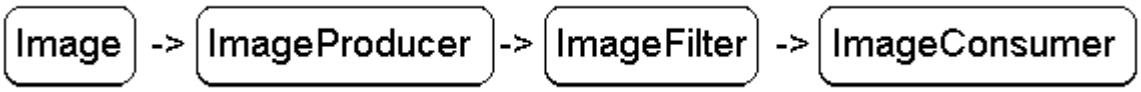
IndexColorModel(int, int, byte[], byte[], byte[])

Muodostaaa IndexColorModel-olion kolmesta byte-tyyppisestä taulukosta, jotka kertovat R, G ja B -komponenttien arvot. Parametrit: väribittien määrä, väritalukon koko, punainen väritaulukko, vihreä väritaulukko ja sininen väritaulukko.

IndexColorModel-luokan voi liittää edelliseen esimerkkiin seuraavasti: käytetään MemoryImageSource-luokan muodostinta, joka tarvitsee parametrikseen värimallin. Pikseleiden arvo pitää olla pienempi kuin väribitit potenssiin 2. Kun väribittien määrä on 8, pikselin arvo saa olla enintään 256. Huomaa ero edelliseen esimerkkiin rivillä, jossa asetetaan pikseleiden arvot.

```
// tee pikselit
for (int x = 0; x < leveys*korkeus; x+=1)
 pikselit[x] =  x%256;

// tee RGB-värikartta
byte[] pun = new byte[256];
byte[] sin = new byte[256];

for (int x = 0; x < 256; x++)
{
 pun[x] = (byte) x;
 sin[x] = (byte) 0;
}

// Tee uusi värimalli
IndexColorModel malli = new IndexColorModel(8, 256,
                                        pun, sin, sin);
```

Suodattimet (*filters*) ovat luokkia, jotka käsittelevät kuvan pikseleitä jollakin tapaa. Niiden avulla voidaan käsitellä myös verkosta ladattuja GIF-kuvia, ei pelkästään muistissa luotuja.



ImageFilter-luokka on perusluokka, joka toimii ImageConsumer- ja ImageProducer-rajapintojen välissä ja antaa valmiudet suodattimien tekoon (luokkaan perustuvia suodattimia on valmiina kaksi kappaletta). ImageFilter-luokka on itse niin kutsuttu nolla-suodatin, joka ei tee mitään.

- CropImageFilter
- RGBImageFilter

Suodatin tarvitsee avukseen **FilteredImageSource**-luokan, joka siis korvaa edellä esitetyn MemoryImageSource-luokan. Luokka on toteutettu ImageProducer-rajapinnasta, joka lukee olemassaolevan kuvan ja suodattimen ja käyttää niitä luomaan uuden suodatetun kuvan alkuperäisestä.

```
Image kuva = createImage(new FilteredImageSource
                         (src.getSource(), colorfilter));
```

FilteredImageSource-luokassa on yksi muodostin. Se tarvitsee parametrikseen ImageProducer-rajapinnasta periytyneen olion, joka saadaan Image-oliosta funktiolla **`getSource`**. Toinen parametri on ImageFilter-luokka tai siitä periytynyt. Luokkakirjastossa niitä on kaksi: RGBImageFilter ja CropImageFilter.

FilteredImageSource(ImageProducer, ImageFilter)
: Muodostaa ImageProducer-olion olemassa olevasta ImageProducer-oliosta ja suodattimesta.

RGBImageFilter on luokka, jonka tarkoituksena on tuottaa helppo tapa luoda ImageFilter-olio, joka muuttaa kuvan pikseleitä yksi kerrallaan käyttäen oletusvärimallia. Luokka on tarkoitettu käytettäväksi yhdessä FilteredImageSource-olion kanssa, jolloin alkuperäisestä kuvasta voidaan suodattaa värejä ja näin synnyttää uusi suodatettu kuva.

RGBImageFilter on myös tarkoitettu periytyväksi luokaksi (se on tyypiltään abstrakti, joten siitä ei voi tehdä oliota). Luokassa on funktio nimeltä **`filterRGB`**, joka mahdollistaa pikselien muuttamisen yksi kerrallaan, riippumatta värimallista, jota ImageProducer käyttää.

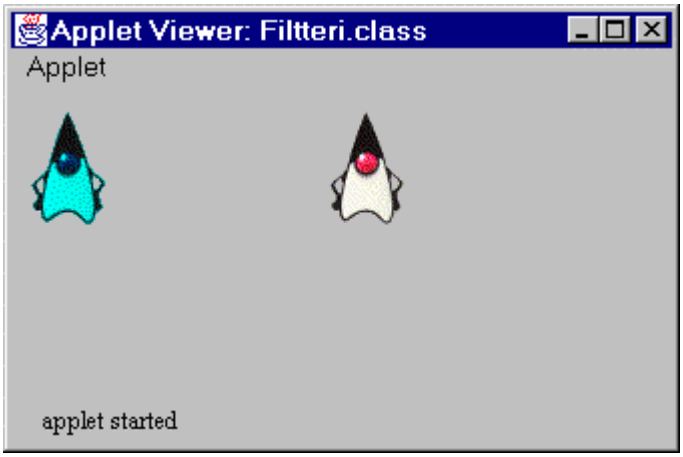
Luokassa on muuttaja nimeltä **`canFilterIndexColorModel`**, joka määrää voiko filterRGB-funktio käsitellä väritaulukon lukuja, kun käytössä on värimalli IndexColorModel. Alaluokan pitää itse kirjoittaa tämän muuttujan arvoksi tosi (true), jos eivät ole riippuvaisia pikselin koordinaateista.

```
    public ColorFilter()
    {
     canFilterIndexColorModel = true;
    }
```

Java-ohjelma Filtteri.java esittelee itse tehtyä suodatinta. Ohjelma koostuu kahdesta luokasta: Filtteri-luokka on appletti, joka ajetaan. ColorFilter-luokka on suodatin, joka suodattaa punaisen värin pois.
Se perustuu yksinkertaisesti riviin,

```
    return ((rgb &0xff00ffff));
```

joka tekee loogisen AND-operaation parametrin rgb ja luvun `0xff00ffff` kanssa. Koska punainen komponentti on nolla, lopputuloksena palautuvat vain värit vihreä ja sininen.




```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;

public class Filtteri extends Applet
{
 Image kuva, lahde;

 public void init()
 {
  lahde = getImage(getDocumentBase(),"Java.gif");
  ImageFilter colorfilter = new ColorFilter();
```

```
  kuva = createImage(new
    FilteredImageSource(lahde.getSource(), colorfilter));
 }

 public void paint(Graphics g)
 {
  g.drawImage(kuva, 1, 1, this);
  g.drawImage(lahde, 150, 1, this);
 }
}

class ColorFilter extends RGBImageFilter
{
  public ColorFilter()
  {
   canFilterIndexColorModel = true;
  }

  public int filterRGB(int x, int y, int rgb)
  {
   return ((rgb &0xff00ffff));
  }
}
```

12.3 Äänet

Javassa on tuki vain **AU**-tiedostolle eli UNIXin äänitiedostolle. Yleisempiä äänitiedostoja, kuten wav Java ei tunne. AU-äänitiedosto on huonompi kuin esimerkiksi wav-tyyppinen. Siinä pakkaustekniikka on heikompi.

Äänien toistamiseen on olemassa kaksi eri tapaa. AU-tiedosto voidaan lukea **`getAudioClip`**-funktiolla. Funktio tarvitsee **AudioClip** nimisen rajapinnan, johon äänitiedosto sisällytetään. Rajapinnassa on funktiot äänitiedoston toistamiseen.

```
AudioClip audio;
```

Parametrikseen getAudioClip-funktio tarvitsee URL-osoitteen ja äänitiedoston nimen, ja mahdollisen hakupolun. Omassa ohjelmassa perus-URL-osoitteeksi kannattaa laittaa funktio **`getCodeBase`** tai **`getDocumentBase`**.

```
audio = getAudioClip(getCodeBase(), ”tiedosto.au”);
```

Äänitiedosto soitetaan AudioClip-rajapinnan funktiolla **`play`**. Rajapinnan muita funktioita ovat **`loop`** ja **`stop`**. Play toistaa äänitiedoston kerran, loop toistaa jatkuvasti ja stop lopettaa.

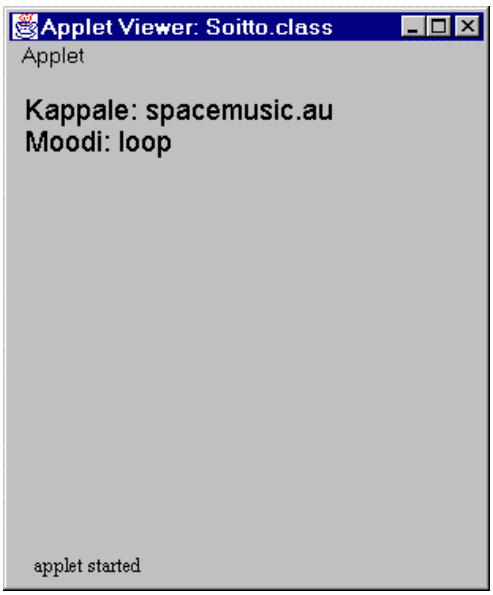
play	Toistaa kerran
loop	Toistaa kokoajan
stop	Lopettaa soittamisen

Applet-luokassa on vielä toinen funktio, joka osaa toistaa äänitiedoston kerran (muuta sillä ei voi tehdä). Funktio on nimeltään **`play`**. Se tarvitsee parametrikseen perus-URL-osoitteen ja äänitiedoston hakupolkuineen.

```
play(getCodeBase(), ”tiedosto.au”);
```

Esimerkki **Soitto.java** on parametrilla ohjattava soitto-ohjelma. Parametrit, jotka ohjaavat soitto-ohjelman toimintaa ovat:

kappale	Toistettavan äänitiedoston nimi
moodi	Toisto moodi: play tai loop




```
import java.awt.*;
import java.applet.*;

public class Soitto extends Applet
{
 AudioClip audio;
```

```
 Thread th;
 String kappale;
 String moodi;

 public void init()
 {
  setFont(new Font("Arial", Font.PLAIN, 18));
  kappale = getParameter("kappale");
  moodi = getParameter("moodi");
  audio = getAudioClip(getDocumentBase(), kappale);
 }

 public void start()
 {
  if (moodi.equals("play"))
   audio.play();
  if (moodi.equals("loop"))
   audio.loop();
 }

 public void paint(Graphics g)
 {
  g.drawString("Kappale: "+kappale, 10, 30);
  g.drawString("Moodi: "+moodi, 10, 50);
 }

 public void stop()
 {
  th = null;
  audio.stop();
 }
}
```

12.4 Animaatio

Animaatio on nykypäivän kuuma sana (varsinkin Java-ohjelmoinnissa), tosin koneiden huima kehitys on tuonut sen rinnalle myös videon. Videossa on kuitenkin se vika, että se vaatii suuren kapasiteetin niin tietokoneelta kuin siirtotieltäkin. Animaatio puolestaan kuluttaa kapasiteettia huomattavasti vähemmän, mutta samalla elävän videon tuottama aitous katoaa.

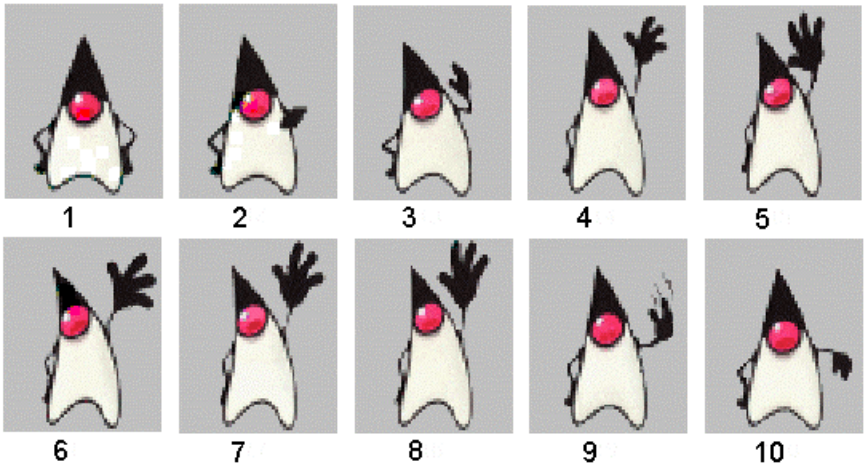
Animaatiota voi toteuttaa usealla eri tavalla. Tässä tutustutaan kahteen eri tapaan. Ensiksi samoista kuvista tehtyyn animaatioon ja sen jälkeen peräkkäisistä kuvista tehtyyn animaatioon.

Jo muutamalla kuvalla voi toteuttaa yksinkertaista animaatiota siirtelemällä kuvia eri paikkoihin. Esimerkiksi ihmishahmo voitaisiin muodostaa kuudesta osasta: päästä, käsistä, keskivartalosta ja jaloista. Kun näitä osia liikuttaa paikasta toiseen, saadaan tunne liikkuvasta ihmisestä. Tässä

tavassa on etuna se, että käyttäjän ei tarvitse imuroida pitkiä kuvatiedostoja, koska osat voidaan esimerkiksi piirtää grafiikkafunktioilla. Vektorigrafiikkaa käyttäen päästään todella upeisiin suorituksiin, mutta Javan ”hitaus” pitää ottaa huomioon.

Toisaalta osat voi ladata valmiina kuvina Internetistä, jolloin huononkaan piirtäjän ei tarvitse hävetä kuviaan. Kun sitten osia liikutetaan suhteessa toisiinsa saadaan taas animaatiota.

Hyvä animaatio on sitä, että kuvia näytetään peräkkäisessä järjestyksessä siten, että syntyy tunne liikkuvasta kuvasta. Lisäksi hahmoa voidaan samanaikaisesti liikuttaa X- ja Y-suunnassa. Samaa tekniikkaa käytetään tietenkin piirretyissä elokuvissa. Oletetaan, että olisi käytettävissä 8-asentoinen kävelevä ihmishahmo. Kun hahmot esitetään peräkkäin samassa kohdassa, näyttää tulos siltä kuin ihmishahmo kävelisi paikallaan. Kun ihmishamoa liikutetaan X-suunnassa jokaisen uuden kuvan näyttämisen välissä, syntyy lopullinen todellinen kävelevä ihmishahmo. Java-ohjelmointipaketin mukana tulee **Animator**-niminen sovellus, joka käyttää tätä tekniikkaa. Animaatiossa esiintyy jo monille tutuksi tullut Duke. Duket on numeroitu 1:stä 10:een, ja ne näytetään peräkkäin.



Tietokoneella animaation esittäminen on helppoa. Luetaan valmiit kuvahahmot muistiin, alustetaan silmukka, jossa kuvia esitetään ja aloitetaan kuvien piirtäminen yksitellen. Kuvien näyttämisen välissä pitää olla viivettä, koska muutoin silmin havaittavaa animaatiota ei synny. Hyvä kuvien esitysnopeus on 24 kuvaa sekunnissa, jolloin syntyy aitoa liikettä. Viive on tällöin 40 millisekuntia. Seuraavassa on esitetty animaation idea pääpiirteissään.

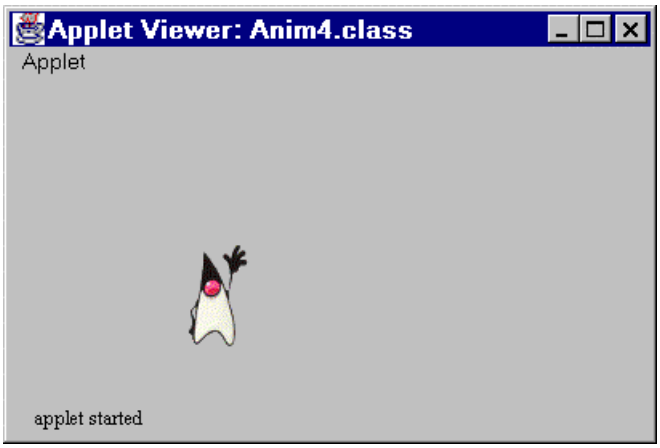
- pyyhitään vanha kuva

- päivitetään koordinaatit
- piirretään seuraava kuva
- viive (40 - 100 ms)

Yllä on esitetty kaikki tarpeelliset kohdat, jotka animaatiosilmukan on toteutettava. Vanhan kuvan pyyhkiminen toteutetaan esimerkiksi **`clearRect`**-funktiolla (huomaa, että koordinaatteja ei saa vaihtaa ennen pyyhintää). Sitten piirretään uusi hahmo uusiin koordinaatteihin, suoritetaan viive **`sleep`**-funktiolla ja jatketaan silmukkaa jälleen alusta.

```
public void init()
{
 // lue parametrit
 // lataa kuvat
}

public void start()
{
  // Aloita thread-prosessi
}

public void run()
{
  // Päivitä piirtopinta
  // viive
  // Päivitä piirtopinta
  // jne.
}

public void paint(Graphics g)
{
 // päivitä koordinaatit
 // Piirrä kuva
 // vaihda kuva
}
```

Esimerkki **anim1.java** esittelee käytännössä animaation tuottamista peräkkäisillä kuvilla. Esimerkki ei tunne parametreja, joten kuvat ladataan tietystä hakemistosta.




```
/*

   Animoi peräkkäisiä kuvia

*/
import java.awt.*;
import java.applet.*;

public class Anim1 extends Applet implements Runnable
{
 Image kuvat[];      // Animaatioesityksen kuvat
 int x = 100, y = 100, kuva = 0; // Kuvan koordinaatit
 Thread th;          // thread

 public void init()
 {
  // Varaa kuvataulukko
  kuvat = new Image[10];

  // Lue kuvat muistiin
  for (int i = 0; i < 10; i++)
  kuvat[i] =
         getImage(getCodeBase(),"images/T"+(i+1)+".gif");
 }

 public void start()
 {
  // Aloita säie
  if (th == null) {
   th = new Thread(this);
   th.start();
  }
 }
```

```
 public void run()
 {
  while (true)
  {
   // Päivitä piirtopinta
   repaint();

   // viive
   try {
    th.sleep(100);
   } catch (InterruptedException e) { }
  }
 }

 public void paint(Graphics g)
 {
  // piirrä kuva
  g.drawImage(kuvat[kuva],x,y,this);
  // Vaihda uusi kuva
  kuva++;
  if (kuva == 10)
   kuva = 0;
 }

 public void stop()
 {
  if (th != null)
   th.stop();
  th = null;
 }
}
```

12.5 Värinätön animaatio

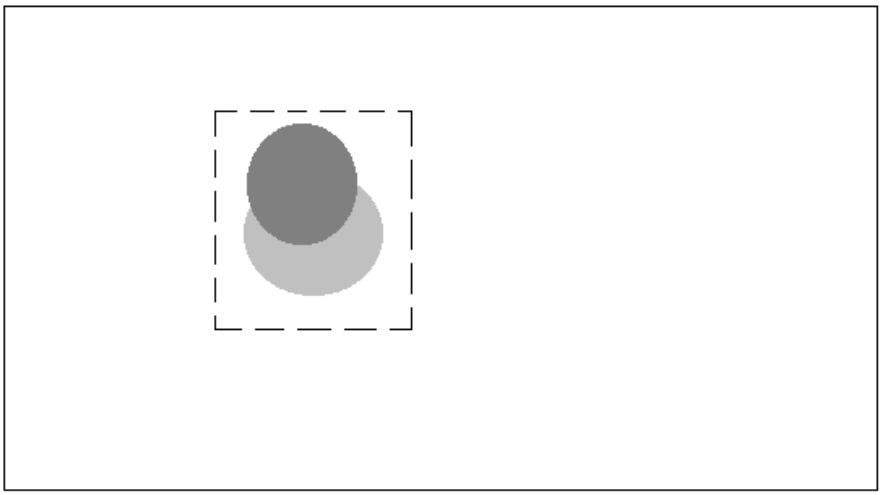
Edellinen animaatio värisi aika lailla. Eikä pelkästään animaatio värise, vaan kaikki peräkkäiset piirtopinnalle suoritetut operaatiot. Vika siis ei ole Javassa vaan tavassa, jolla operaatiot suoritetaan. Tähän mennessä on käytetty paint-funktiota päivittämään piirtopinnalle grafiikkaa. Samalla näkymättömänä funktiona on toiminut **`update`**, joka maalaa piirtopinnan taustavärillä eli tyhjentää piirtopinnan.

- update-funktio maalaa koko piirtopinnan taustavärillä
- paint-funktio päivittää grafiikan

Yksi tapa on valjastaa update-funktio omaan tarkoitukseen. Kun update-funktio ylikirjoitetaan, se ei enää tyhjennetä piirtopintaa. Voidaan kirjoittaa rutiini, joka tyhjentää vain tarvittavan osan -osan, johon animaatio on piirretty. Tällä tavoin grafiikan esittämisestä tulee vähemmän välkkyvää.

- update-funktio maalaa koko piirtopinnan taustavärillä
- paint-funktio päivittää grafiikan

Esimerkiksi alla olevassa kuvassa ei kannata tyhjentää koko piirtopintaa (ehyt suorakulmio). Järkevämpää on tyhjentää piirtopinta vain kuvion kohdalta (katkoviivalla esitetty alue).



```
public void paint(Graphics g)
{
 update(g);
}

public void update(Graphics g)
{
 g.clearRect(100,100,100,100);
 g.copyArea(100,100, 50,50,200,200);
}
```

Seuraavaksi kirjoitetaan animaatio-ohjelman runko:

```
public void init()
```

```
    {
     // lue parametrit
     // lataa kuvat
    }

    public void start()
    {
      // Aloita thread-prosessi
    }

    public void run()
    {
      // Päivitä piirtopinta
      // viive
      // Päivitä piirtopinta
      // jne.
    }

    public void update(Graphics g)
    {
     // lue kuvan korkeus ja leveys
     // Nollaa piirtoalue kuvan kohdalta
     // Vaihda kuva
     // päivitä koordinaatit
     // Piirrä kuva
    }
```

Esimerkki **anim2.java** esittelee animaation tuottamista update-funktion avulla. Esimerkki perustuu edelliseen, eikä siten tunne parametreja.

```
/*

   Animoi peräkkäisiä kuvia
   Käyttää värinän poistamiseen update-funktiota
*/
import java.awt.*;
import java.applet.*;

public class Anim4 extends Applet implements Runnable
{
 Image kuvat[];          // Animaatioesityksen kuvat
 int x = 100, y = 100, kuva = 0;// Kuvan koordinaatit
 Thread th;          // thread
 int leveys, korkeus;
```

```
public void init()
{
 // Varaa kuvataulukko
 kuvat = new Image[10];

 // Lue kuvat muistiin
 for (int i = 0; i < 10; i++)
  kuvat[i] =
       getImage(getCodeBase(),"images/T"+(i+1)+".gif");
}

public void start()
{
 // Aloita säie
 if (th == null) {
  th = new Thread(this);
  th.start();
  }
 }

public void run()
{
 while (true)
 {
  // Päivitä piirtopinta
  repaint();

  // viive
  try {
   th.sleep(1000);
  } catch (InterruptedException e) { }
 }
}

public void update(Graphics g)
{
 // Lue edellisen kuvan korkeus ja leveys
 leveys = 20 + kuvat[kuva].getWidth(this);
 korkeus =20 + kuvat[kuva].getHeight(this);

 // Pyyhi vanha kuva piirtopinnalta
 g.clearRect(x,y,leveys,korkeus);

 // Vaihda uusi kuva
 kuva++;
 if (kuva == 10)
  kuva = 0;
```

```
   // piirrä kuva
   g.drawImage(kuvat[kuva],x,y,this);
  }

  public void stop()
  {
   if (th != null)
    th.stop();
    th = null;
  }
 }
```

12.6 Kaksoispuskurointi

Kaksoispuskurointi on paras tapa yhdessä update-funktion kanssa toteuttaa värinätön animaatio. Kaksoispuskurointi tarkoittaa sitä, että on olemassa kaksi erillistä piirtopintaa, joihin vuorotellen piirretään. Ne voitaisiin nimetä lyhyesti piirtopinnoiksi 1 ja 2. Alla on esitetty järjestyksessä kaikki ne toimenpiteet, jotka kaksoispuskurointi tarvitsee.

1. Näytetään piirtopinta 1
2. Piirretään piirtopinnalle 2
3. (Viive esimerkiksi 40ms - 100ms)
4. Näytetään piirtopinta 2
5. Piirretään piirtopinnalle 1
6. (Viive esimerkiksi 40ms - 100ms)

Idea on yksinkertainen: piirtopinta, johon piirretään, on näkymättömissä. Kun kaikki piirto-operaatiot on suoritettu, vaihdetaan kyseinen piirtopinta näkyviin (viive ei ole pakollinen). Tällöin ei voi syntyä värinää, koska näkyvissä olevalle piirtopinnalle ei piirretä.

Kaksoispuskurointia ei voi tässä muodossa käyttää Javassa, koska ei ole funktioita piirtopinnan vaihtamiseen. Eikä ole funktiota, jolla voitaisiin kopioida toiselta piirtopinnalta toiselle. Ongelman voi kiertää, käyttämällä hieman toisenlaista tekniikkaa: toiseksi piirtopinnaksi valitaan kuva. Image-luokka sisältää oman piirtopinnan ja funktion getGraphics. Image-luokan avulla voidaan hyödyntää kaksoispuskurointia seuraavalla tavalla:

- Luodaan tyhjä kuva
- Luetaan kuvan piirtopinta getGraphics-funktiolla
- Piirretään kuva piirtopinnalle
- Näytetään kuva drawImage-funktiolla oikealle piirtopinnalle

Ohjelmointikaan ei ole hirveän iso urakka. Tyhjän kuvan luominen tapahtuu Component-luokan **`createImage`**-funktiolla. Piirtopinnan mitat saadaan saman luokan **`bounds`**-funktiolla.

```
Image ruutu = createImage(korkeus,leveys);
```

Piirtopinta luetaan nyt **`getGraphics`**-funktiolla:

```
Graphics g1 = ruutu.getGraphics();
```

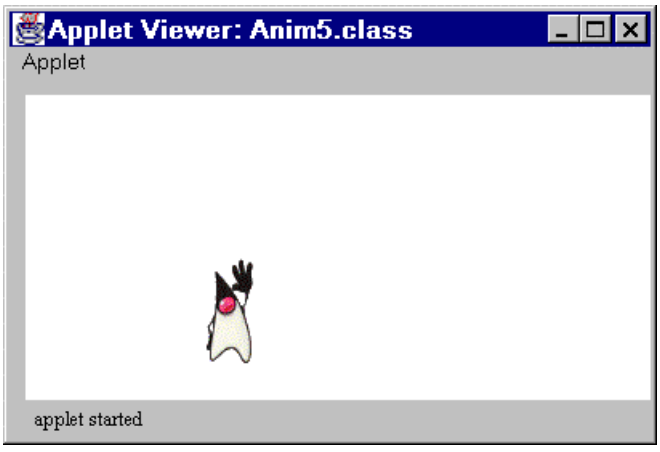
Piirtäminen tapahtuu saadulle piirtopinnalle:

```
g1.drawRect(10,10,100,100);
```

Lopuksi kuvan piirtopinta piirretään oikealle piirtopinnalle. Käytännössä se tapahtuu piirtämällä kuva **`drawImage`**-funktiolla.

```
g.drawImage(ruutu, x, y, this);
```

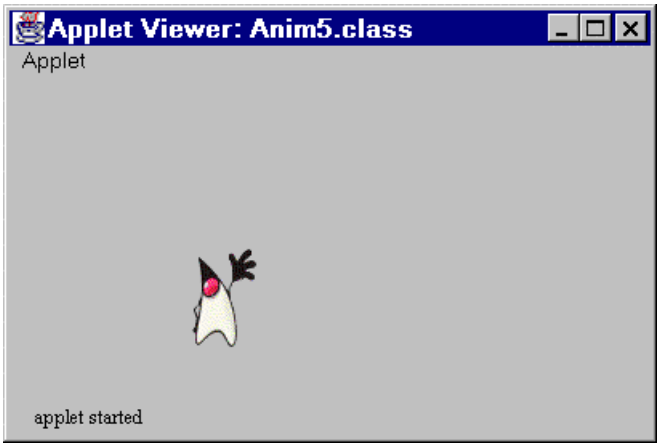
Vielä kaksoispuskurointi ei ole täydellinen. Itse tehdyssä kuvassa pohjaväri on tietenkin erilainen kuin appletilla. Kun kuvan tulostaa piirtopinnalle, tulos on jotakin tällaista:



Lisätään vielä yksi ohjelmarivi, jolla kuvan piirtoväri muutetaan samaksi kuin appletin taustaväri. Piirtoalueen nollaamiseen tarkoitettu clearRect-funktio muutetaan fillRect-funktioksi:

```
// Aseta piirtoväri
```

```
g1.setColor(getBackground());

// Pyyhi vanha kuva piirtopinnalta
g1.fillRect(0, 0, 100, 100);
```



Viimeisenä tehtävänä on estää kuvien näyttäminen ennen kuin animaamaatio esitys todella alkaa. Olet saattanut huomata tämän siten, että kuvat appletin käynnistyessä piirtyvät hitaasti ja epätasaisesti.

- Estetään kuvien ennen aikainen näyttäminen ylikirjoittamalla ImageObserver-rajapinnan imageUpdate-funktio

```
public boolean imageUpdate(Image i, int info, int x,
                           int y, int width, int height)
{
 return true;
}
```

Esimerkki **anim3.java** esittelee käytännössä animaation tuottamista update-funktion ja kaksoisbufferoinnin avulla. Esimerkki perustuu edelliseen eikä siten tunne parametreja. Tämäkään applet ei ole täysin värinätön ja se johtuu siitä, että piirtopinnalle piirtämistä ei ole tahdistettu monitorin kanssa. Eikä sitä voi myöskään Javalla toteuttaa, joten parhaat värinättömät appletit saadaan, kun

- päivitetään vain tietty alue update-funktiolla
- käytetään kaksoispuskurointia

```
/*

   Animoi peräkkäisiä kuvia (anim3.java)
   Käyttää värinän poistamiseen update-funktiota ja
   kaksoispuskurointia.
   Kuvien ennen aikainen esiintyminen on estetty
   ylikirjoittamalla imageUpdate-funktio.

*/
import java.awt.*;
import java.applet.*;

public class Anim3 extends Applet implements Runnable
{
 Image kuvat[];     // Animaatioesityksen kuvat
 int x = 100, y = 100, kuva = 0; // Kuvan koordinaatit
 Thread th;         // thread
 int leveys, korkeus;
 Image ruutu;
 Graphics g1;
  public void init()
  {
   // Tee piirtopinta 2
   ruutu = createImage(100, 100);
   g1 = ruutu.getGraphics();
   g1.setColor(getBackground());

   // Varaa kuvataulukko
   kuvat = new Image[10];

   // Lue kuvat muistiin
   for (int i = 0; i < 10; i++)
    kuvat[i] =
        getImage(getCodeBase(),"images/T"+(i+1)+".gif");
  }

  public void start()
  {
   // Aloita säie
   if (th == null) {
    th = new Thread(this);
    th.start();
    }
   }

  public void run()
  {
```

```
   while (true)
   {
    // Päivitä piirtopinta
    repaint();

    // viive
    try {
     th.sleep(100);
    } catch (InterruptedException e) { }
   }
  }

  public void update(Graphics g)
  {
   // piirrä kuvan (ruutu) piirtopinnalle
    // Lue edellisen kuvan korkeus ja leveys
    leveys = 20 + kuvat[kuva].getWidth(this);
    korkeus = 20 + kuvat[kuva].getHeight(this);

    // Pyyhi vanha kuva piirtopinnalta
    g1.fillRect(0, 0, 100, 100);

    // Vaihda uusi kuva
    kuva++;
    if (kuva == 10)
     kuva = 0;

    // piirrä kuva
    g1.drawImage(kuvat[kuva], 1, 1, this);
   // päivitä kuva oikealle piirtopinnalle
    g.drawImage(ruutu, x, y, this);
  }


  public boolean imageUpdate(Image i, int info, int x,
                      int y, int width, int height)
  {
   return true;
  }

  public void stop()
  {
   if (th != null)
    th.stop();
    th = null;
  }
}
```

Seuraava esimerkki esittelee TekstiDemon kaltaisen parametreilla ohjattavan animaattorin **AnimaatioDemo**n. AnimaatioDemoa on parannettu ensimmäisestä painoksesta. Se sisältää nyt muun muassa kaksoisbufferoinnin.

```
/*

    AnimaatioDemo By Petri Kiuttu & Juha Niemi  1996

    Appletin parametrit:

    suorita , jossa toiminnot
     V - kulje vasemmalle
     O - kulje oikealle
     Y - kulje ylös
     A - kulje alas
     E - esitys paikallaan
     S - pysäytä
     U - Seuraava kuvasarja
     F - lopeta
     C - aloita alusta

    sarja1-16   Kuvasarjat
    kuvia       Kuvien määrä kuvasarjoissa
    nopeus      Esitysnopeus
*/

import java.awt.*;
import java.applet.*;
import  java.net.URL;
import java.lang.*;

public  class  AnimaatioDemo  extends  Applet  implements
Runnable
 {
  Image kuvat[][];
  String suorita;
  int x=0,y=0;  // Kuvan koordinaatit
  Thread th;// thread-prosessi

  // Muuttujat varattu eri toiminoille
  int count=0;
  boolean loppu=true;
  int kuva, kuvia = 18, hahmo;
  int komento;
  int time;
```

```
  int nopeus=3;
  char merkki;
  int kasky;
  Font ft;
  int suunta;

  Image ruutu;
  Graphics g1;

  public void init()
  {
   // Tee piirtopinta 2 kaksoispuskurointia varten
   ruutu         =         createImage(getBounds().width,
getBounds().height);
   g1 = ruutu.getGraphics();
   g1.setColor(getBackground());

   // Lue parametrit suorita, nopeus, nopeus ja kuvia
   suorita = getParameter("suorita");
   try {
        if (getParameter("nopeus")!=null)
         nopeus = Integer.valueOf(getParameter("nopeus"))
                                                .intValue();
   } catch (NumberFormatException e) {}
   try {
        kuvia = Integer.valueOf(getParameter("kuvia"))
                                             .intValue();
   } catch (NumberFormatException e) {}

   // lue parametrit sarja1-16 ja alusta kuvat
   String nimi;
   kuvat = new Image[16][kuvia+1];
   for (int i=0; i<16; i++)
   {
     nimi = getParameter("sarja"+(i+1));
     if (nimi==null)
      break;
     for (int e=0; e<=kuvia; e++)
       kuvat[i][e] = getImage(getCodeBase(),
                              nimi+e+".gif");
   }
  }

  public void start()
  {
   // Aloita prosessin suorittaminen
   if (th == null)
```

```
 {
   th = new Thread(this);
   th.start();
 }
}

public void stop()
{
 th.stop();
 th = null;
}

public void run()
{
 // Alusta muuttujat
 count=0;
 kasky=0;
 kuva=0;
 hahmo=0;
 loppu=true;

 // Aseta X- ja Y-koordinaatit
 x=getBounds().width/2;
 y=getBounds().height/2;

 // Suorita päälooppia, kunnes F (FINISH)
 // tai C (CONTINUE) jatkaa alusta
 while (loppu)
 {
  count=0; time=0;
  if (kuvat[kuva][0] == null)
    kuva=0;

  // Tee komennosta tokeni
  merkki = suorita.charAt(kasky);
  if (merkki == 'V') komento=3;
  if (merkki == 'O') komento=4;
  if (merkki == 'Y') komento=5;
  if (merkki == 'A') komento=6;
  if (merkki == 'S') komento=7;
  if (merkki == 'E') komento=8;
  if (merkki == 'U') {kuva += 1; kasky += 2; continue;}
  if (merkki == 'F') {loppu = false; continue; }
  if (merkki == 'C') {kasky = 0; continue; }

  try {   // Tee suoritusaika
```

```
      time = Integer.parseInt(suorita.substring(
                              kasky+1,kasky+4), 10);
   } catch (NumberFormatException n) { }
  kasky+=5;     // Osoita seuraavaa käskyä

  // Suorita toiminto
  for (int i=1; count<time; i++)
  {
   if (komento!=7)// Jos ei ole STOP-komento
   {
    // nolla hahmo, jos seuraavaa ei ole
    hahmo+=1;
    if (hahmo>kuvia)
     hahmo=0;
    repaint();
   }
   else
    count+=nopeus;

   // Viivettä
   try {
        th.sleep(120);
   } catch (InterruptedException e) { }
  }
 }
 komento=1;
}

public void paint(Graphics g)
{
 update(g);
}

public boolean imageUpdate(Image i, int info, int x,
                           int y, int width, int height)
{
 return true;
}

public void update(Graphics g)
{
 // Pyyhi vanha kuva ruudun piirtopinnalta
 g1.fillRect(0, 0, ruutu.getWidth(this),
                 ruutu.getHeight(this));

 if (komento==1)
 {
```

```
      g1.drawImage(kuvat[kuva][hahmo],x,y,this);
      return;
    }

    count+=nopeus;
    switch (komento)
    {
      case 3:    // Vieritä vasemmalle
        x-=nopeus;
        break;

      case 4:    // Vieritä oikealle
        x+=nopeus;
        break;

      case 5:    // Vieritä ylös
        y-=nopeus;
        break;

      case 6:    // Vieritä alas
        y+=nopeus;
        break;

      case 8:    // Esitys paikallaan
        break;
    }
    // piirrä kuva ruudulle
    g1.drawImage(kuvat[kuva][hahmo], 10, 10,this);

    // päivitä lopuksi ruutu oikealle piirtopinnalle
    g.drawImage(ruutu, x, y, this);
  }
}
```

12.7 Multimediaa

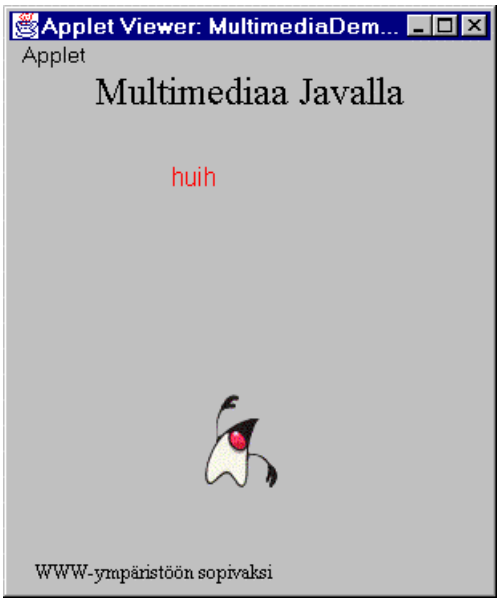
Multimedia sisältää ääntä ja kuvaa (tai liikkuvaa kuvaa). Kuten on jo nähty, Javalla on erinomaiset mahdollisuudet toteuttaa huomiota herättäviä esityksiä omille kotisivulle. Loppuluvun aikana tutustutaan yhden esimerkkiohjelman avulla multimedian tuottamiseen. Se on yksinkertainen esimerkki, jossa ääntä, animaatiota ja tekstiä tuotetaan samanaikaisesti. Hyvä multimedia esitys on myös JDK:n mukana tullut appletti Animator. Siihen kannattaa tutustua.

Multimedia esityksessä eri osat pitäisi saada tahdistettua toisiinsa: niitä ovat ääni, kuva ja teksti. On siten mahdollista tuottaa esitys, joka esitetään äänien mukaan tai kuvien mukaan. Äänen tahdistaminen muihin osiin on paljon vaikeampaa kuin toisinpäin, koska ääniominaisuudet ovat sen verran alkeelliset.

MultimediaDemo.java on appletti, jossa säie tahdistaa kaikki osat. Run-funktiossa kutsutaan paint-funktiota, joka demossa vastaa kaikkien osien näyttämisestä. Paint-funktion tehtävät ovat:

- piirtää kuva
- soittaa musiikkia
- tulostaa tekstiä

Appletissa soitetaan vain yksi äänitiedosto, mutta helposti voitaisiin jokaisen kuvan kohdalla soittaa sille ominaninen ääni. Tietenkään tällöin animaatioesityksen nopeus ei voi olla kovin suuri, koska äänet pitää ehtiä soittamaan .




```
/*

   Multimedia-esitys (multi.java)
   Käyttää värinän poistamiseen update-funktiota ja
   kaksoispuskurointia

*/
import java.awt.*;
import java.applet.*;

public class MultimediaDemo extends Applet implements
Runnable
{
  int x = 100, y = 100, kuva = 0;// Kuvan koordinaatit
 int leveys, korkeus, kuvia = 17, aloita = 0;
```

```
 int count = 0;
 Image ruutu;
 Graphics g1;
 Color vari = Color.red;

 Font font1;
 int counter;
 Thread th;

 // Multimedia esityksen komponentit
 AudioClip aanet;
 Image kuvat[];          // Animaatioesityksen kuvat
 String teksti[] =
 {
  "Katsoppas tätä", "huh huh", "tässä sitä mennään",
  "eikä meinata!", "Moro", "olen Duke", "ystävät kutsuu",
  "minua sillä nimellä", "tässä sitä mennään", "huih",
  "jaha!", "on aika taas", "alkaa alusta", "pyörähtely",
  "oli kiva tavata", "moido!", ""
 };

 String status[] =
 {
  "Duke multimedia esitys",
  " on toteuttu Java-kielellä",
  "WWW-ympäristöön sopivaksi",
  "Toivottavasti nautit siitä"
 };

  public void init()
  {
   // aseta fontti
   font1 = new Font("TimesRoman",Font.PLAIN,24);
   setFont(font1);

   // Tee piirtopinta 2 kaksoispuskurointia varten
   ruutu         =         createImage(getBounds().width,
getBounds().height);
   g1 = ruutu.getGraphics();
   g1.setFont(new Font("Arial",Font.PLAIN,16));

   // Lue kuvat muistiin
   kuvat = new Image[kuvia];
   for (int i = 0; i < kuvia; i++)
    kuvat[i] =
     getImage(getCodeBase(),"multimedia/T"+(i+1)+".gif");
```

```
   // Lue äänet muistiin
   aanet =
    getAudioClip(getCodeBase(),"multimedia/T1.au");
  }

  public void start()
  {
   if (th == null)
   {
    th = new Thread(this);
    th.start();
   }
  }

  public void run()
  {
   while (true)
   {
    // Tulosta statusteksti
    showStatus(status[(counter%16)/4]);
    // Tulosta animaatio, esitysteksi. Soita äänet
    repaint();
    try
    {
     th.sleep(600); // Tulosta kuvia viiveellä
    } catch (InterruptedException e) {}
   }
  }

  public void update(Graphics g)
  {
   paint(g);
  }

  public void paint(Graphics g)
  {
   // piirrä kuvan (ruutu) piirtopinnalle
   // Lue edellisen kuvan korkeus ja leveys
   leveys = 20 + kuvat[counter%kuvia].getWidth(this);
   korkeus = 20 + kuvat[counter%kuvia].getHeight(this);

   // Pyyhi vanha kuva piirtopinnalta
   g1.setColor(getBackground());
   g1.fillRect(0,          0,          getBounds().width,
getBounds().height);

   // Uusi esitys alkaa
```

```
   counter++;

   // soita äänet
   if (counter%kuvia == 1)
    aanet.play();

   // piirrä kuva
   g1.setColor(vari);
   g1.drawString(teksti[counter%kuvia],
                 getBounds().width/3,70);
   g1.drawImage(kuvat[counter%kuvia],
getBounds().width/3,
getBounds().height - 100, this);

   // päivitä kuva oikealle piirtopinnalle
   g.drawImage(ruutu, 1, 1, this);

   // tulosta tekstiä (keskitä)
   String m = "Multimediaa Javalla";
   int x = (getBounds().width -
             getFontMetrics(font1).stringWidth(m))/2;
   g.drawString("Multimediaa Javalla", x, 20);
 }


  public void stop()
  {
   if (th != null)
    th.stop();
    th = null;
  }
}
```

13. Esimerkkiohjelmia

Tässä luvussa on Java-kielisiä esimerkkiohjelmia. Esimerkit löytyvät CD-ROMilta valmiina appletteina HTML-sivulla. CD:llä on myös ohjelmien lähdekoodit.

13.1 Othello

Pelissä on mahdollisuus kaksinpelaamiseen, joka osaltaan lisää mielenkiintoa. Pelissä on myös seuraavat alla olevat toiminnot, jotka voidaan kytkeä päälle tai pois.

- Pelinappulan sijoituspaikat (Toiminnot->Risti)
- Tietokonepelaaja (Toiminnot->Tietokone)

Tarkoituksena pelissä on "syödä" vastustajan pelinappuloita, ja saada siten mahdollisimman paljon omia pelinappuloita. Voittaja on se, jolla lopussa on pelinappuloita eniten.

- Pelaajalla 1 ovat punaiset pelinappulat (aloittaa pelin)
- Pelaajalla 2 ovat siniset pelinappulat
- Tietokone on pelaaja 2 (tietokonepelaaja aktivoidaan valikkokomennolla toiminnot->tietokone)

Pelaaja asettaa pelinappulan siten, että kahden oman pelinappulan välissä on vähintään yksi vastustajan pelinappula (syönti tapahtuu kaikkiin kahdeksaan ilmansuuntaan). Tietokone tarkastaa pelaajien siirrot ja ilmoittaa virheellisistä siirroista viestilaatikolla (pelin helpottamiseksi Toiminnot-valikosta voidaan valita valikkokomento Risti, joka näyttää mihin pelinappula on mahdollista asettaa).

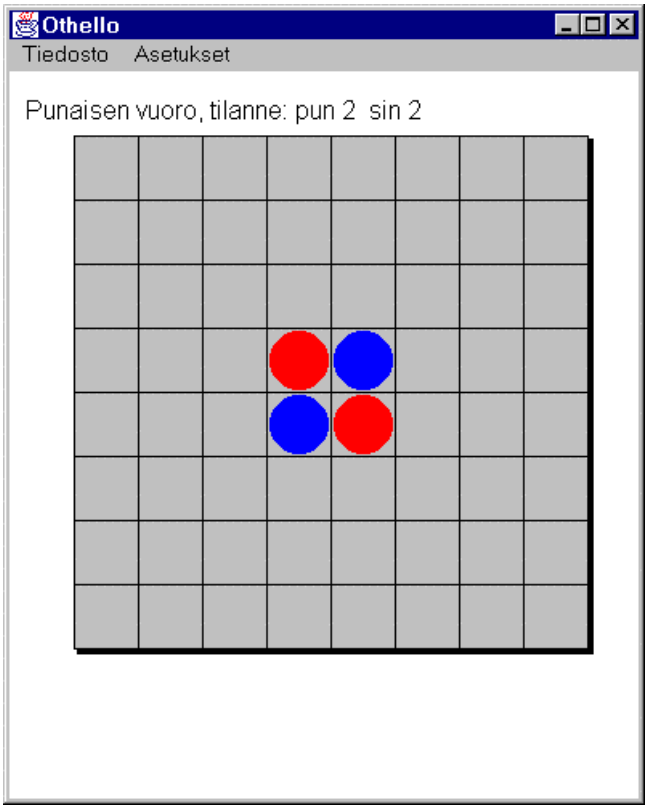
Othello voidaan käynnistää joko Java-tulkissa, appletviewer-ohjelmalla tai tietenkin selaimella.

Othello.html-tiedosto on seuraavanlainen:

```
<HTML>
<BODY>
<TITLE> Othello </TITLE>
<H1><CENTER> Othello - kaksinpeliä Internetissä
</CENTER></H1><P>
<H3> Othello peli aukeaa erilliseen ikkunaan. Sulkeminen tapahtuu
menukomennolla Tiedosto->Lopeta</H3>
```

```
<APPLET code=Othello.class>
</APPLET>
</BODY>
</HTML>
```

Kuvassa Othello on ajettu Java-tulkilla.




```
/*
     OTHELLO V 1.0
     Ohjelmoinut Petri Kiuttu
     Windows-ympäristöön 1995
     Java-ympäristöön 1996

     OTHELLO.JAVA   (Tämä tiedosto)
     OTHELLO.CLASS (Applet)
     OTHELLO.ICO (Ikoni)
     OTHELLO.HLP (Helppitiedosto)

*/
```

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.net.*;
import java.io.*;

public class Othello extends Applet
{
 Frame f;

 public void init()
 {
  f = new OthelloIkkuna("Othello");
 }

 public void stop()
 {
  f.setVisible(false);
  f.dispose();
 }

 public static void main(String args[])
 {
  new OthelloIkkuna("Othello");
 }

}

class OthelloIkkuna extends Frame implements Runnable,
ActionListener, MouseListener, MouseMotionListener
{
 Color RED = Color.red;
 Color BLUE = Color.blue;

 int RdkX, RdkY, RdkSize;
 int pelialue[] = new int[100];
 int vuoro = 0, computer = 0, risti = 0;
 int oldx, oldy;
 int dialogi, pelaaja;
 String merkkijono;
 String pelaaja1;
 String pelaaja2;

 Rectangle rc;
 Point pt;
 int x, y, X, Y, i, max, oldmax;
```

```
String tuloste;
MenuBar menubar;
Menu menu;
MessageBox d;

public OthelloIkkuna(String nimi)
{
 // Alusta peli
 uusi();
 vuoro = 1;
 pelaaja = 1;

 // aseta ikkunan tiedot
 setLayout(new CardLayout());
 setTitle(nimi);
 setCursor(new Cursor(DEFAULT_CURSOR));
 setBackground(Color.white);
 setSize(400, 600);
 setFont(new Font("Arial", Font.PLAIN, 16));
 setResizable(false);

 // aseta menu
 MenuItem mi;
 menubar = new MenuBar();
 menubar.add (menu = new Menu("Tiedosto"));
 menu.add (mi = new MenuItem("Uusi"));
 mi.addActionListener(this);
 menu.add (mi = new MenuItem("Tietoa"));
 mi.addActionListener(this);
 menu.addSeparator();
 menu.add (mi = new MenuItem("Lopeta"));
 mi.addActionListener(this);
 menubar.add (menu = new Menu("Asetukset"));
 menu.add (mi = new MenuItem("Risti"));
 mi.addActionListener(this);
 menu.add (mi = new MenuItem("Tietokone"));
 mi.addActionListener(this);
 setMenuBar(menubar);

 addMouseListener(this);
 addMouseMotionListener(this);

 // näytä ikkuna
 setVisible(true);
 //show();
 setSize(400, 500);
}
```

```
 public void paint(Graphics g)
 {
  // Lue ikkunan leveys & korkeus
  if (getBounds().width < getBounds().height)
   RdkSize = getBounds().width / 10;
  else
   RdkSize = getBounds().height / 10;
  RdkX = (getBounds().width - (8 * RdkSize)) / 2;
  RdkY = 80;
  // Piirrä pelikenttä
  rc = new Rectangle(RdkX, RdkY, RdkX + (8 * RdkSize),
RdkY + (8 * RdkSize));
  g.setColor(Color.lightGray);
  g.fillRect(rc.x, rc.y, 8 * RdkSize, 8 * RdkSize);

  // Piirrä varjostus
  g.setColor(Color.black);
  g.fillRect(rc.x + 2, rc.height, 2 + (8 * RdkSize), 4);
  g.fillRect(rc.width, rc.y + 2, 4, (8 * RdkSize));

  // Piirrä ruudukot
  g.clearRect(0, 0, getBounds().width, 39);
  g.drawString("Punaisen vuoro", 10, 30);
  y = RdkY;
  for (i = 0; i <= 8; i++) {
    g.drawLine(RdkX, y, RdkX + (RdkSize * 8), y);
    y += RdkSize;
   }
  x = RdkX;
  for (i = 0; i <= 8; i++) {
    g.drawLine(x, RdkY, x, RdkY + (RdkSize * 8));
    x += RdkSize;
   }

  // Piirrä pelinappulat ruudulle
  X = 0; Y = 0;
  x = 1; y = 1;
  for (i = 0; i <= 63; i++)
  {
   if (pelialue[i] == 1)
    X++;
   if (pelialue[i] == 2)
    Y++;
   if (pelialue[i] != 0)
    PiirraPallo(x, y, pelialue[i]);
   if (x>=8)
```

```
   {
    x = 0;
    y++;
   }
   x++;
  }

  // Tulosta ikkunaan tietoa
  if (vuoro == 1 && pelaaja1 == null)
    merkkijono = "Punaisen vuoro, ";
  if (vuoro == 2 && pelaaja2 == null)
    merkkijono = "Sinisen vuoro, ";
  if (vuoro == 2 && computer == 1)
    merkkijono = "Tietokoneen vuoro, ";
  g.clearRect(0, 0, getBounds().width, 39);
  g.drawString(merkkijono+"tilanne: pun "+X+"  sin "+Y,
10, 30);
 }

 public void mousePressed(MouseEvent e)
 {
  int x = e.getX();
  int y = e.getY();
  if (d != null)
   if (d.isVisible() == true)
    return;
  x = 1 + ((x - RdkX) / RdkSize);
  y = 1 + ((y - RdkY) / RdkSize);
  pelivuoro(x, y);
 }
 public void mouseClicked(MouseEvent e) { }
 public void mouseReleased(MouseEvent e) { }
 public void mouseEntered(MouseEvent e) { }
 public void mouseExited(MouseEvent e) { }


 public void mouseMoved(MouseEvent e)
 {
  int x = e.getX();
  int y = e.getY();

  if (risti == 0)
  {
   setCursor(new Cursor(DEFAULT_CURSOR));
   return;
  }
  x = 1 + ((x - RdkX) / RdkSize);
```

```
 y = 1 + ((y - RdkY) / RdkSize);
 // Tutki onko kursorin sijainti muuttunut
 if (x != oldx || y != oldy)
 {
  oldx = x;
  oldy = y;
  if (TutkiSuunnat(x, y, vuoro, 2) == 0)
   setCursor(new Cursor(DEFAULT_CURSOR));
  else
   setCursor(new Cursor(CROSSHAIR_CURSOR));
 }
}

public void mouseDragged(MouseEvent e) { }


public void actionPerformed(ActionEvent e)
{
 // Tutki menukomennot
 // UUSI
  if (e.getActionCommand().equals("Uusi"))
  {
   uusi();
   repaint();
   // Jos tietokone pelaa ensin, aloita peli
   if (computer == 1 && pelaaja == 2)
     pelivuoro(1,1);
  }

 // LOPETA
  if (e.getActionCommand().equals("Lopeta"))
  {
   setVisible(false);
   this.dispose();
  }

 // RISTI
  if (e.getActionCommand().equals("Risti"))
  {
   risti ^= 1;
  }

 // TIETOKONE
  if (e.getActionCommand().equals("Tietokone"))
  {
   computer^=1;  // Vaihda tietokokone / pelaaja2
   // Uusi peli
```

```
    uusi();
    repaint();
    // Jos tietokone pelaa ensin, aloita peli
    if (computer == 1 && pelaaja == 2)
     pelivuoro(1,1);
   }

  // TIETOA
   if (e.getActionCommand().equals("Tietoa"))
   {
    new MessageBox(this,"Tietoa","Ohjelmoinut Petri
        Kiuttu 1996",1).avaa();
   }
 }


 /*

   Tietokone pelaajan säie
 */
 public void run()
 {
  pelivuoro(1,1);
 }

 /*

  Funktio pelivuoro suorittaa pelivuoron vaatimat
toimenpiteet

 */
 void pelivuoro(int x, int y)
 {
  if (vuoro == 0)
   return;

  // Tutki pelivuoro (pelaaja1, pelaaja2, tietokone)
  if (computer == 1 && vuoro == 2)
  {
   tietokone();
  }
  else if (TutkiSuunnat(x, y, vuoro, 1) == 0)
  {
   new MessageBox(this, "Virhe!", "Et voi asettaa tähän",
     1).avaa();
   return;
  }
```

```
  // Vaihda pelivuoroa
  vuoro = (vuoro == 1) ? 2 : 1;

  // Tutki kuinka paljon pelaajilla on pelinappuloita
  X = 0; Y = 0; max = 0;
  for (i = 0; i <= 63; i++)
  {
   if (pelialue[i] == 1)
    X++;
   if (pelialue[i] == 2)
    Y++;
   if (pelialue[i] == 0)
    max++;
  }

  // Tulosta ikkunaan tietoa
  if (vuoro == 1 && pelaaja1 == null)
   merkkijono = "Punaisen vuoro, ";
  if (vuoro == 2 && pelaaja2 == null)
   merkkijono = "Sinisen vuoro, ";
  if (vuoro == 2 && computer == 1)
   merkkijono = "Tietokoneen vuoro, ";
  getGraphics().clearRect(0, 0, getBounds().width, 39);
  getGraphics().drawString(merkkijono+"tilanne: pun "+X+"
     sin "+Y, 10, 30);
  // onko peli päättynyt?
  if (X == 0 || Y == 0 || max == 0)
  {
   new MessageBox(this, "Loppu!", "Peli päättyi "+X+" :
    "+Y, 1).avaa();
   vuoro = 0;
   return;
  }

  // Tutki onko pelaajalla mahdollisuuksia asettaa
nappulaa
  x = 1; y = 1; max = 0;
  for (i = 0; i <= 63; i++)
  {
   max += TutkiSuunnat(x, y, vuoro, 2);
   if (x >= 8)
   {
    x = 0;
    y++;
   }
   x++;
```

```
  }
  if (max == 0 && computer == 0)
  {
   new MessageBox(this, "Vuoro siirtyy!", "Et voi asettaa
    pelinappulaa", 1).avaa();
   vuoro = (vuoro == 1) ? 2 : 1;
  }
  setCursor(new Cursor(DEFAULT_CURSOR));

  // Aloita tietokoneen vuoro, jos tietokone aktivoitu
  if (computer == 1 && vuoro == 2)
   new Thread(this).start();
  }

  /*

    Funktio tietokone suorittaa tietokonepelaajan
toiminnot

  */
  public void tietokone()
  {
   // Tutki mikä vaihtoehdoista on "paras"
   X = 0; x = 1; y = 1; max = 0; oldmax = 0;
   for (i = 0; i <= 63; i++)
   {
    if (pelialue[i] == 0)
     X++;
    max = TutkiSuunnat(x, y, vuoro, 2);
     if (max >= oldmax)
    {
     oldmax = max;
     pt = new Point(x,y);
    }
    if (x >= 8)
    {
     x = 0;
     y++;
    }
    x++;
   }

  // Tutki voiko tietokone asettaa pelinappulaa
  if (oldmax == 0 || X == 0)
   if (X != 0)
   {
```

```
    new MessageBox(this, "Vuoro siirtyy!", "En voi aset
        taa pelinappulaa", 1).avaa();
    return;
   }

 // Aseta pelinappula
 TutkiSuunnat(pt.x, pt.y, vuoro, 1);
}


/*

 Funktio uusi alustaa pelialueen

*/
void uusi()
{
 vuoro = 1;
 if (pelaaja == 2)
  vuoro = 2;

 for (int x = 0; x < 100; x++)
  pelialue[x] = 0;

 pelialue[27] = 1;
 pelialue[28] = 2;
 pelialue[35] = 2;
 pelialue[36] = 1;
}

/*

 Funktio PiirraPallo
  - asettaa pelinappulan

*/

int PiirraPallo(int x, int y, int vari)
{
 Graphics g = getGraphics();
 int X, Y;

 if (x < 1 || y < 1 || x > 8 || y > 8)
  return 0;

 x--;
 y--;
```

```
 if (vari == 1)    // Punainen
  g.setColor(RED);
 if (vari == 2)    // Sininen
  g.setColor(BLUE);

 // Tallenna pelikarttaan nyk. pallon väri
 pelialue[(y * 8) + x] = vari;

 X = RdkX + (x * RdkSize) + 2;
 Y = RdkY + (y * RdkSize) + 2;
 g.fillOval(X, Y, RdkSize - 4, RdkSize - 4);
 return 1;
}

// Tutkii pelinappulan sijoituspaikan oikeellisuuden
int TutkiSuunnat(int x, int y, int pelaaja, int mode)
{
 /*
   MODE 1 - Normaali
   MODE 2 - Tarkistus
 */
 int X, Y, i;
 int pos, vari = 0, error = 1;
 int suunta, counter = 0, max = 0;
 Point maalattavat[] = new Point[64];

 if (x < 1 || y < 1 || x > 8 || y > 8)
  return 0;

 pos = (8 * (y - 1)) + (x - 1);
 if (pelialue[pos] != 0)
  return 0;

 if (pelaaja == 1)
  vari = 2;
 if (pelaaja == 2)
  vari = 1;

 for (suunta = 1; suunta <= 8; suunta++)
 {
  counter = 0;
  X = x; Y = y;
  // Etsi vastustajan pelinappulat
  do
  {
   switch (suunta)
```

```
    {
      case 1:        // Pohjoinen
        Y--;
        break;

      case 2:        // Koilinen
    X++;
    Y--;
    break;

      case 3:        // Itä
    X++;
    break;

      case 4:        // Kaakko
    X++;
    Y++;
    break;

      case 5:        // Etelä
    Y++;
    break;

      case 6:        // Lounas
    X--;
    Y++;
    break;

      case 7:        // Länsi
    X--;
    break;

      case 8:        // Luode
    X--;
    Y--;
     }
     if (X > 0 && X < 9 && Y > 0 && Y < 9)
     {
      pos = (8 * (Y - 1)) + (X - 1);
      // Tallenna maalattavat pelinappulat
      maalattavat[counter] = new Point(X, Y);
     }
     else
       pos = 65;
     counter++;
}
```

```
  while (pelialue[pos] == vari && pos < 65);     // X > 0
&& X < 9 && Y > 0 && Y < 9);

  // Piirrä pelinappulat
  if (pelialue[pos] == pelaaja && counter > 1 && X > 0 &&
X < 9 && Y > 0 && Y < 9)
  {
   max += counter - 1;
   if (mode == 1)
   {
    PiirraPallo(x, y, pelaaja);
    for (i = 0; i < counter; i++)
     PiirraPallo(maalattavat[i].x, maalattavat[i].y,
pelaaja);
   }
  }
 }
 return max;
 }
}
```

13.2 Kello

Kello on appletti, joka näyttää paikallisen kellonajan. Kellon näyttö perustuu java.util-luokkakirjaston Date-luokan funktioihin. Lisäksi apuna käytetään säiettä, jolla tehdään sekunnin mittainen viive.

Tiedoston Kello.html sisältö CD:llä on seuraava:

```
<HTML>
<HEAD>
<TITLE> Kello </TITLE>
</HEAD>
<BODY>
<CENTER> <H1> Kello on nyt: </H1>
<APPLET code=Kello.class
 width=300
 height=100>
</APPLET>
</CENTER>
</BODY>
</HTML>
```

Tiedoston Kello.java sisältö CD:llä on seuraava:

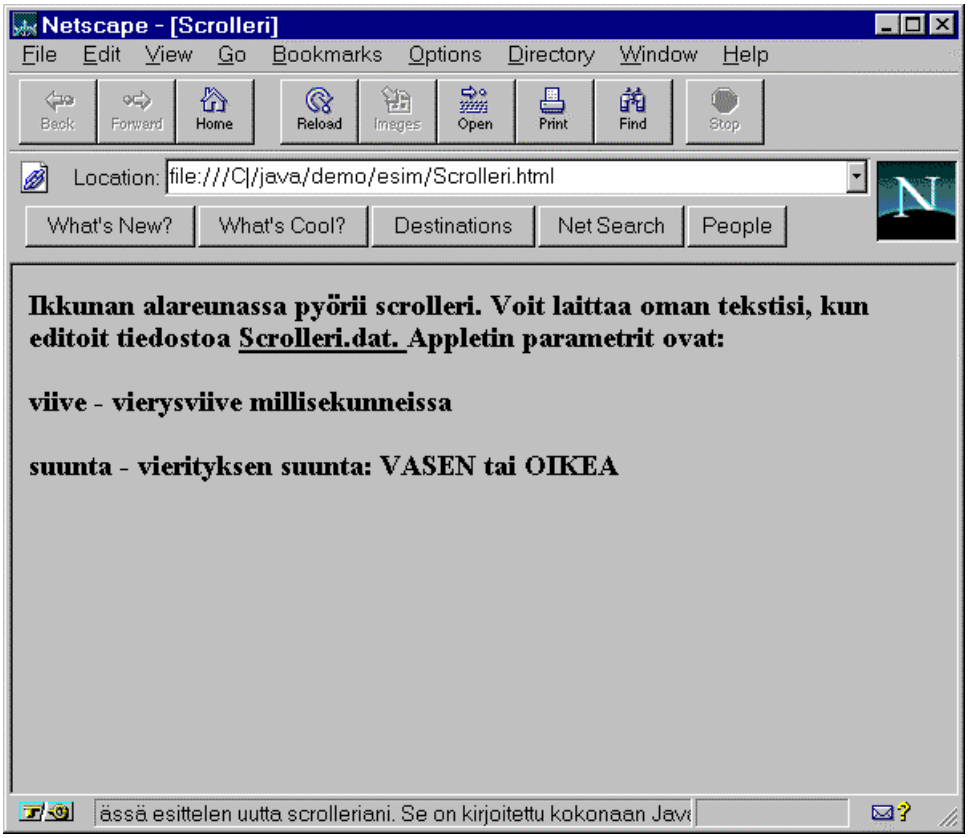
```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;

public class Kello extends Applet implements Runnable
{
 Thread th;
 FontMetrics ft;
 int leveys, korkeus, laskuri;

 public void init()
 {
  // aseta fontti
  Font font = new Font("Times New Roman", Font.BOLD, 36);
  setFont(font);
  ft = getFontMetrics(font);
  leveys = getBounds().width;
  korkeus = getBounds().height;
 }

 public void start()
 {
```

```
  if (th == null)
  {
   th = new Thread(this);
   th.start();
  }
 }

 public void run()
 {
  while (true)
  {
   repaint();
   laskuri++;
   try
   {
    th.sleep(1000);
   }
   catch (InterruptedException e) {}
  }
 }

 public void paint(Graphics g)
 {
  // tee aika
 Calendar cal = Calendar.getInstance();

  // Joka viides sekunti näytetään päivämäärä
  int fleveys;
  String aika = "";
  if ((laskuri%6) == 5)
  {
   // Näytä päivämäärä
   aika = DateFormat.getDateInstance().format(new
Date());
   fleveys = ft.stringWidth(aika);
  }
  else
  {
   // Näytä kellonaika
   aika = DateFormat.getTimeInstance().format(new
Date());
   fleveys = ft.stringWidth(aika);
  }

  // Keskitä ja tulosta kellonaika tai päivämäärä

  int x = (leveys-fleveys)/2;
```

```
  int y = korkeus/2;
  g.drawString(aika, x, y);
 }

 public void stop()
 {
  th.stop();
  th = null;
 }
}
```

13.3 Scrolleri

Scrolleri on parametreilla ohjattava appletti, joka vierittää annettua tekstiä tilapalkissa. Kuvassa on appletti Scrolleri.java, joka on tehty tätä kirjaa varten. Tekstin näyttö perustuu java.applet-luokkakirjaston Applet-luokan **`showStatus`**-funktioon.



Scrolleri-appletti tuntee kaksi parametria. Niiden merkitys on seuraava:

viive vieritysviive millisekunneissa. Oletus: 400

suunta vierityksen suunta: VASEN tai OIKEA.
Oletus: VASEN

Vieritysteksti annetaan tiedostossa **Scrolleri.dat**. Tiedoston pitää olla samassa hakemistossa kuin appletin koodi. Teksti alkaa tiedoston alusta ja päättyy rivinvaihtoon. Tekstin alussa voidaan käyttää välilyöntejä erottamaan tekstin alku ja loppu vieritysvaiheessa. Tiedoston Scrolleri.dat sisältö voi olla seuraava:

```
Heippa! Tässä esittelen uutta scrolleriani. <ENTER>
```

Tiedoston **Scrolleri.html** sisältö voi olla seuraava:

```
<HTML>
<HEAD>
<TITLE> Scrolleri </TITLE>
</HEAD>
<BODY>
<APPLET code=Scrolleri.class
 width=10
 height=10>
<PARAM name="viive" value="100">
<PARAM name="suunta" value="OIKEA">
</APPLET>
</BODY>
</HTML>
```

Tiedoston Scrolleri.java sisältö CD:llä on seuraava:

```
import java.awt.*;
import java.applet.*;
import java.net.*;
import java.io.*;

public class Scrolleri extends Applet implements Runnable
{
 String teksti = "";
 String vieritys;
 Thread th;
```

```
 int viive = 400;
 String suunta ="";

 public void init()
 {
  if (getParameter("viive") != null)
   viive =
Integer.valueOf(getParameter("viive")).intValue();

  if ((suunta = getParameter("suunta")) == null)
   suunta = "VASEN";

  // lue vieritysteksti
  try
  {
   URL osoite = new URL (getDocumentBase(),
    "scrolleri.dat");
   InputStream input = osoite.openStream();
   BufferedReader data = new BufferedReader(new
    InputStreamReader(input));
   // Lue teksti readLine()-funktiolla
   teksti = data.readLine();
   } catch (MalformedURLException em) {}
     catch (IOException ei) {}
 }

 public void start()
 {
  if (th == null) {
   th = new Thread(this);
   th.start();
  }
 }

 // Vieritys tapahtuu säikeessä
 public void run()
 {
  if (suunta.equals("VASEN"))   // Vieritä oikealta
vasemmalle
  {
    // Lue merkkijonon pituus
    int x = 0;
    // aseta aloitusmerkkijono
    vieritys = teksti;
    x++;
    // Vierityssilmukka alkaa tästä
    while (true)
```

```
    {
     // tulosta vierivä teksti
     showStatus(vieritys);
     // poista eka merkki ja liitä merkkijono loppuun
     vieritys = teksti.substring(x,teksti.length()) +
        teksti.substring(0, x-1);
     // Seuraava merkki
     x++;
     if (x == teksti.length())
     x = 1;
     // Generoi viivettä
     try
     {
      th.sleep(viive);
     } catch (InterruptedException e) {}
    }
  }

  if (suunta.equals("OIKEA"))   // Vieritä vasemmalta
oikealle
  {
    // Lue merkkijonon pituus
    int x = teksti.length();
    // aseta aloituskirjain
    vieritys = teksti.substring(x-1,x);
    x--;
    // Vierityssilmukka alkaa tästä
    while (true)
    {
     // tulosta vierivä teksti
     showStatus(vieritys);
     // lisää seuraava merkki rivin alkuun ja
     // liitä vierivä merkkijono merkin perään
     vieritys = teksti.substring(x-1,x) + vieritys;
     // Seuraava merkki
     x--;
     if (x == 0)
     x = teksti.length();
     // Generoi viivettä
     try
     {
      th.sleep(viive);
     } catch (InterruptedException e) {}
    }
  }
 }
}
```

Liite A. JDK:n työkalut

Tässä liitteessä esitellään Java Developer Kitin mukana tulevat työkalut.

javac - Java-kääntäjä

Javac on Java-kääntäjä, joka kääntää Java-ohjelmia. Komento **javac_g** on ei-optimoitu versio javac-kääntäjästä, jota käytetään debuggerien kanssa.

Yhteenveto

javac [options] filename.java ...
javac_g [options] filename.java ...

Kuvaus

Komento **javac** kääntää ascii-muotoisen lähdekoodin tavukoodeiksi. Sen jälkeen käytetään Java-tulkkia (java-komento) tulkitsemaan tavukoodit tietokoneen ymmärtämälle konekielelle.

Lähdekoodin pitää olla tiedostossa, jonka pääte on **.java** (esimerkiksi koodi.java). Jokainen luokka joka on määritelty lähdekoodissa syötetään javac-kääntäjälle, jolloin kääntäjä tallentaa tavukoodit tiedostoon, jonka pääte on **.class** (esimerkiksi koodi.class). Kääntäjä asettaa .class-tiedostot samaan hakemistoon kuin missä .java-tiedosto sijaitsee (ellei käytetä -d optiota).

Kun määrittelet omia luokkia, kääntäjälle tarvitsee kertoa niiden sijainti. Käytä siihen CLASSPATH-optiota. CLASSPATH koostuu puolipisteillä erotetusta hakemisto listasta, joka määrittelee luokkien sijainnin. Jos lähdekoodissa syötetään javac-kääntäjälle luokka, jota ei ole määritelty missään muussa tiedostossa, silloin javac hakee luokkaa CLASSPATH-option määräämistä paikoista.

.;C:\java\omat\luokat
Huomaa, että järjestelmä liittää aina järjestelmän luokkien sijainnin hakupolun loppuun, ellei käytössä ole CLASSPATH-optio, jolla määritellään täydellinen hakupolku.

Optiot

-classpath path — Määrittelee polun, jota javac käyttää hakupolkuna luokkien etsimiseen. Ylikirjoittaa oletuspolun tai CLASSPATH-ympäristömuuttujan, jos se on asetettu. Hakemistot erotetetaan toisistaan puolipisteillä. Esimerkki:
.;C:\omat\datat\luokat;C:\java\omat\luokat

-d directory — Määrittelee juuripolun luokkien hierarkiassa.
Esimerkiksi komento

javac -d <;oma_hakemisto>; Ohjelma.java
aiheuttaa sen, että .class-tiedostot, jotka tulevat Ohjelma.java käännöksestä, tallennetetaan hakemistoon oma_hakemisto

-g Sallii kääntäjän tuottaa debug-taulukot.
Debug-taulukot sisältävät tietoa rivinumeroista ja paikallisista muuttujista. Näitä tietoja hyödyntää Java debug-työkalu jdb. Oletuksena vain rivinumerot tuotetaan ellei optiota O käännetä päälle.

-nowarn Poistaa varoitukset. Jos tätä optiota käytetään, kääntäjä ei tulosta yhtään varoitusta.

-O Optimoi käännetyn koodin sisällyttäen
staattiset, lopulliset ja yksityiset funktiot. Huomaa, että luokkien koko saattaa suurentua.

-verbose Aiheuttaa sen, että kääntäjä ja linkkeri tulostavat
tietoa siitä mitä lähdekoodia käänneteään ja mitä luokkia ladataan.

Ympäristömuuttujat

CLASSPATH Käytetään tuottamaan järjestelmälle polku, jossa
sijaitsevat käyttäjän määrittelemät luokat. Esimerkiksi:
.;C:\omat\datat\luokat;C:\java\omat\luokat

java - Java-tulkki

Java-tulkki tulkitsee (suorittaa) tavukoodin. Komento **javac_g** on optimoimaton versio Java-tulkista, jota käytetään debuggerien kanssa.

Yhteenveto

java [optiot] luokan nimi <;argumentit>;
java_g [optiot] luokan nimi <;argumentit>;

Kuvaus

Komento **java** suorittaa Java-tavukoodin, jonka Java-kääntäjä javac on tuottanut.

Luokan nimiargumentti on luokan nimi, joka suoritetaan. Luokan nimi pitää olla kelvollinen, esimerkiksi sisältäen paketin nimen.

```
java java.lang.String
```

Huomaa, että mikä tahansa argumentti, joka esiintyy luokan nimen jälkeen komentorivillä, syötetään main-funktiolle parametriksi.

Tulkki olettaa luokan tavukoodin sijaitsevan tiedostossa, jonka pääte .class. Oikean päätteinen tiedosto tuotetaan automaattisesti, kun lähdekoodi käännetään javac-kääntäjällä. Luokan pitää sisältää myös main-funktio ennen kuin se voidaan ajaa Java-tulkilla. Funktio määritellään seuraavasti:

```
class luokka
{
 public static void main(String argv[])
 {
 }
}
```

Tulkki suorittaa main-funktion ja sen jälkeen poistuu ellei luoda säikeitä. Jos yksi tai useampi säie on luotu, funktiosta ei poistuta ennen kuin viimeinenkin säie loppuu.

Kun määrittelet omia luokkia, sinun tarvitsee kertoa niiden sijainti. Käytä siihen CLASSPATH-optiota. CLASSPATH koostuu puolipilkuilla erotetuista hakupoluista. Esimerkiksi:

```
;C:\javasorsa\luokat
```

Huomaa, että järjestelmä liittää aina järjestelmän luokkien sijainnin hakupolun loppuun, ellei käytössä ole CLASSPATH-optio, jolla määritellään täydellinen hakupolku.

Tavallisesti käännät lähdekoodit Java-kääntäjällä ja sen jälkeen ajat ohjelmat Java-tulkilla. Kuinka tahansa, Java-tulkkia voi käyttää myös kääntämään lähdekoodin ja sen jälkeen suorittamaan tavukoodit. Se onnistuu, kun käytät optiota **-cs**. Tulkki testaa ennen kääntämistä onko luokka ajan tasalla - ellei ole lähdekoodi käännetään.

Tulkki tarkastaa onko luokka laillinen. Tarkastaminen varmistaa, että tavukoodit jotka suoritetaan, eivät riko turvallisuusmääräyksiä.

Optiot

-debug	Sallii debug-ohjelman (jdb) liittyä tähän istuntoon. Kun -debug on annettu komentorivillä, tulkki tulostaa tunnussanan, jota pitää käyttää aloitettaessa debug-toiminto.
-cs	Kun käännetty luokka on ladattu, tämä optio aiheuttaa luokan uudelleen kääntämisen, jos luokka on vanhempi kuin lähdekoodi. Vertaaminen tapahtuu päivämäärien perusteella.
-classpath path	Määrittelee polun, jota javac käyttää hakupolkuna luokkien etsimiseen. Ylikirjoittaa oletuspolun tai CLASSPATH-ympäristömuuttujan, jos se on asetettu. Hakemistot erotetetaan toisistaan puolipisteillä. Esimerkki: .;C:\omat\datat\luokat;C:\java\omat\luokat
-mx x	Asettaa muistinvarauksen maksimikoon. Oletus on 16 megatavua. HUOM! Koon pitää olla vähintään 1000 tavua.
-ms x	Asettaa muistinvarauksen aloituskoon. Oletus on 1 megatavu. HUOM! Koon pitää olla vähintään 1000 tavua.
-noasyncgc	Asettaa pois tahdistamattoman automaattisen muistinvapautuksen.
-ss x	Jokaisella säikeellä on kaksi pinoa: yksi Java-koodille ja yksi C-koodille. Optio -ss asettaa maksimikoon pinolle, jota käytetään C-koodissa. Koon pitää olla vähintään 1000 tavua.
-oss x	Jokaisella säikeellä on kaksi pinoa: yksi Java-koodille ja yksi C-koodille. Optio -oss asettaa pinon maksimikoon Java-koodille. Koon pitää olla vähintään 1000 tavua.
-t	Tulostaa koodit, jotka jäljitetään (vain java_g)
-v, -verbose	Tulostaa jokaisen luokan, joka ladataan
-verify	Tarkastaa jokaisen koodin
-verifyremote	Tarkastaa jokaisen koodin, joka on ladattu järjestelmään classloaderin kautta. Tämä on oletusasetus.
-noverify	Estää tarkastamisen
-verbosegc	Tulostaa viestin aina kun muistia vapautetaan automaattisesti
-DpropertyName =newValue	Määrittää uudelleen ominaisuuden. PropertyName on ominaisuuden nimi, joka vaihdetaan annettuun arvoon. Esimerkiksi:

```
java -Dawt.button.color=green ...
```

Ympäristömuuttujat

CLASSPATH	Käytetään tuottamaan polku, jossa sijaitsee käyttäjän määrittelemät luokat. Esimerkiksi: ;C:\omat\datat\luokat;C:\java\omat\luokat

jdb - Java-debuggeri

Java-debuggeri auttaa löytämään ja korjaamaan virheet.

Yhteenveto

```
jdb [ optiot ]
```

Kuvaus

Komento **jdb** on dbx-tyylinen komentorivi debuggeri. Se käyttää apuna Java Debugger APIa.

On olemassa kaksi tapaa käyttää debuggeria. Yleisin tapa on aloittaa Java-tulkilla luokan suoritus ja debuggaus yhtä aikaa. Se tehdään kirjoittamalla komento jdb ja luokan nimi. Esimerkiksi:

```
C:\> jdb browser.hotjava
```

Kun aloitetaan tällä tavalla, jdb käyttää apuna Java-tulkkia. Luokka ladataan, ja tulkkaaminen pysäytetään ennen kuin yhtäkään tavukoodia suoritetaan.

Toinen tapa on käyttää debug-ohjelmaa liittymään Java-tulkkiin, joka on jo käynnissä. Turvallisuussyistä Java-tulkki sallii debuggauksen ainoastaan, jos debug-optiota on käytetty. Kun tulkki käynnistetään tällä optiolla, tulostetaan salasana, jota tarvitaan jdb:ssä. Esimerkiksi:

```
C:\> jdb -host <;isäntä>; -salasana <;salasana>;
```

Optiot

HUOM! Kun debuggeria käytetään Java-tulkkina, se hyväksyy samat optiot kuin java.

Kun debuggeria käytetään liittymällä Java-tulkkiin. Se tunnistaa seuraavat optiot:

-host <;hostname>	Asettaa isäntäkoneen nimen, jonka

istuntoon liitytään.

-password <;password>; Liittyy istuntoon tällä tunnussanalla.

Ympäristömuuttujat

CLASSPATH Käytetään tuottamaan polku, jossa sijaitsee käyttäjän määrittelemät luokat. Esimerkiksi: C:\omat\datat\luokat;C:\java\omat\luokat

javap - Disassembler

Disassembler purkaa luokan tavukoodit

Yhteenveto

javap [optiot] luokka

Kuvaus

Komento **javap** purkaa luokan tavukoodin. Tuloste riippuu optioista. Jos optioita ei käytetä, tulostetaan luokan yleiset muuttujat ja funktiot. Tulostus ohjataan stdout-tietovirtaan.

Esim. Kun tehdään alla oleva koodi, ja syötetään se javap-ohjelmalle.

```
class C {
    static int a = 1;
    static int b = 2;
    static {
        System.out.println(a);
    }
    static {
        a++;
        b = 7;
        System.out.println(a);
        System.out.println(b);
    }
    static {
        System.out.println(b);
```

```
        }
        public static void main(String args[]) {
            C c = new C();
        }
    }
```

Tulos (ei optioita):

```
Compiled from C:\users\dac\C.java
private class C extends java\lang\Object {
   static int a;
   static int b;
   public static void main(java\lang\String []);
   public C();
   static void ();
}
```

Optiot

-l Tulostaa rivitaulukon ja paikallisen muuttuja taulukon

-p Tulostaa private - ja protected-tyyppiset funktiot ja yleiset muuttujat.

-c Tulostaa puretun koodin.

-classpath path Määrittelee polun, jota javac käyttää hakupolkuna luokkien etsimiseen. Ylikirjoittaa oletuspolun tai CLASSPATH-ympäristömuuttujan, jos se on asetettu. Hakemistot erotetetaan toisistaan puolipisteillä. Esimerkki: ;C:\omat\datat\luokat;C:\java\omat\luokat

Ympäristömuuttujat

CLASSPATH Käytetään tuottamaan polku, jossa sijaitsee käyttäjän määrittelemät luokat. Esimerkiksi: ;C:\omat\datat\luokat;C:\java\omat\luokat

javadoc - API-dokumentin tuottaja

Tuottaa API-dokumentteja lähdekoodeista

Yhteenveto

javadoc [optiot] paketti | tiedosto.java

Kuvaus

Komento **javadoc** erottaa määrittelyt ja erityiset dokumentti kommentit lähdekoodista ja muotoilee niistä API:n HTML-formaattiin. Argumenttina voit antaa, joko paketin nimen tai listan lähdekoodeista.

Dokumenttikommenteissa javadoc tukee myös erityisiä komentoja joilla parannetaan API-dokumentaatiota. Javadoc tukee myös standardia HTML-kieltä kommenteissa.

Paketti, joka annetaan komentorivillä, pitää olla määriteltynä CLASSPATH-optiossa tai -ympäristömuuttujassa. Huomaa, että javadoc käyttää .java-päätteisiä tiedostoja eikä .class-päätteisiä.

Javadoc muotoilee ja näyttää kaikki yleiset ja suojatut määrittelyt:

- Luokissa ja rajapinnoissa
- Funktioissa
- Muuttujissa

Dokumentti kommentit

Lähdekoodiin voidaan lisätä dokumentti kommentti. Kommentti alkaa merkillä /** ja välissä on teksi, joka lisätään dokumenttiin.

Standardi HTML-kieli

Voit lisätä standardeja html-kielen tageja dokumentti kommentin sisään. Älä kuitenkaan käytä otsikko tageja kuten <;h1>; tai <;hr>;. Koska javadoc luo kokonaisen rakenteen dokumenttiin ja nämä rakenne tagit häiritsevät dokumentin muotoilua.

Javadoc-ohjelman dokumenttikomennot

Javadoc eristää erityiset komennot, jotka se tunnistaa dokumentti kommentin välissä. Komennot sallivat automaattisesti tuottaa täydellisen, hyvin muotoiltun APIn lähdekoodeista. Tagit alkavat merkillä @.

Komennon pitää alkaa rivin alusta. Pidä samannimiset komennot yhdessä dokumenttikomennon sisässä. Esimerkiksi laita kaikki @author (=kirjoittaja) komennot yhteen, jotta javadoc voi kertoa missä lista loppuu

@see classname
@see fully-qualified-classname
@see fully-qualified-classname#method-name
@version version-text
@author your-name

Optiot

-d directory Määrittelee hakemiston johon dokumentit generoidaan. Esimerkiksi:
javadoc -d C:\usrs\dac\public_html\doc java.lang

-verbose Tulostaa lähdekoodit, jotka käännetään ja oliot, jotka ladataan.

-classpath path Määrittelee polun, jota javac käyttää hakupolkuna luokkien etsimiseen. Ylikirjoittaa oletuspolun tai CLASSPATH-ympäristömuuttujan, jos se on asetettu. Hakemistot erotetetaan toisistaan puolipisteillä. Esimerkki:
;C:\omat\datat\luokat;C:\java\omat\luokat

Ympäristömuuttujat

CLASSPATH Käytetään tuottamaan polku, jossa sijaitsee käyttäjän määrittelemät luokat. Esimerkiksi:
;C:\omat\datat\luokat;C:\java\omat\luokat

appletviewer - Applettien näyttäjä

Appletviewer mahdollistaa applettien ajamisen ilman WWW-selainta.

Yhteenveto

appletviewer [optiot] url-osoitteet

Kuvaus

Komento **appletviewe**r lataa dokumentin, jonka URL-osoite kertoo. Jokainen appletti esitetään omassa ikkunassaan. HUOM! Jos URL-osoite ei viittaa dokumenttiin, joka sisältää appletteja, niin appletviewer ei tee mitään.

Optiot

-debug Aloittaa applet viewerin debug-ohjelmaa sallien applettien debuggauksen.

Hakemisto

A

Abstract-muunnin; 64
Alimenu; 181, 182
Animaatio; 20
API; 25, 26, 59
Applet-luokka; 103, 145
APPLET-tagi; 100
Appletviewer; 108
Aritmeettiset operaattorit; 48
Arvonmuunto operaattorit; 48
AudioClip-rajapinta; 297
AWT-komponentti; 145

B

Bittitason loogiset operaattorit; 50
Boolean; 38
Boolean-luokka; 80
BorderLayout-luokka; 159
Break-lause; 54
Button-luokka; 207
Byte; 38

C

Canvas-luokka; 238
Cardlayout-luokka; 160
CGI; 15
Character-luokka; 80
CheckboxGroup-luokka; 214
Checkbox-luokka; 210
Choice-luokka; 231
Class; 60
Color-luokka; 261
ColorModel-luokka; 291
Component-luokka; 157
Container-luokka; 103, 149
Continue-lause; 58

D

Datan kätkentä; 58
Date-luokka; 334
Destroy-funktio; 105
Dialog-luokka; 195
DirectColorModel-luokka; 293
Double-luokka; 80
Do-while-lause; 56

E

Elementit; 27
`Event-luokka`; 120
Event-olio; 122, 124
Exception; 82
Extends; 60, 70

F

FileInputStream-luokka; 90
File-luokka; 88
FileOutputStream-luokka; 90
FilteredImageSource-luokka; 295
Float-luokka; 80
FlowLayout-luokka; 163
For-lause; 55
Frame-luokka; 173, 181, 199

G

GIF-kuva; 289
Goto-lause; 58
Graphics-luokka; 260
GridBagConstraints-luokka; 167
GridBagLayout-luokka; 166
GridLayout-luokka; 164
GUI; 103

H

Hajotin; 68
HandleEvent-funktio; 121
Hotjava; 17
HTML-kieli; 7

I

If-else-lause; 52
Ikkuna; 179
ImageConsumer-rajapinta; 295
ImageFilter-luokka; 295
Image-luokka; 289, 307
ImageProducer-rajapinta; 295
Implements; 60, 77
Import; 78
IndexColorModel-luokka; 294
Init-funktio; 103
Insets-luokka; 154, 158
Instanceof-lause; 72
Int; 38
Integer-luokka; 80

J

Java; 15
Java.applet; 79
Java.awt; 78, 79
Java.awt.image; 79
Java.io; 79
Java.lang; 80
Javac-kääntäjä; 29
Java-prosessori; 20
JavaScript; 16, 27
JDK; 22
JPEG-kuva; 289

K

Kapselointi; 58
Kehitysympäristö; 22
keyDown-funktio; 136
`keyUp-funktio`; 136
Komponentti; 149

L

Label-luokka; 228
Layout; 157, 158
List-luokka; 234
Long; 38
Long-luokka; 80
Loogiset operaattorit; 49
Luokka; 58, 60
Luokkakirjasto; 28
Luokkakirjastot; 78

M

Math-luokka; 81
MemoryImageSource-luokka; 294
MemoryImageSource-luokka.; 291
MenuBar-luokka; 181
MenuItem-luokka; 181
Menukomento; 181
Menu-luokka; 181
Menupalkki; 181, 182
Metodi; 26, 58
Moniperiytyvyys; 77
MouseDown-funktio; 139, 140
MouseDrag-funktio; 141
MouseExit; 140
MouseMove-funktio; 141
mouseUp-funktio; 140
Muodostin; 65, 67, 75
Muunnin; 60

N

Native-muunnin; 64
New-operaattori; 46, 65
Nimetty silmukka; 57

O

Object-luokka; 71
Olio; 60
Olio-ohjelmointi; 58

P

`paint-funktio`; 157
Paint-funktio; 104, 106, 260
Panel-luokka; 103, 145
Periytyvyys; 70
Piirtopinta; 259
Private-muunnin; 63
Public-muunnin; 63

R

Radionappi; 213
Rajapinta; 60, 77
RandomAccessFile-luokka; 91
Return-lause; 58
RGBImageFilter-luokka; 295
Runnable-rajapinta; 249, 250

S

Scrollbar-luokka; 224
Short; 38
Sijoitusoperaattori; 48
Sleep-funktio; 301
Sovelluskehitin; 23

Staattinen funktio; 76
Staattinen koodilohko; 76
Staattinen muuttuja; 76
Start-funktio; 104
Stop-funktio; 104
String-luokka; 43
Super-muuttuja; 75
Switch-case-lause; 54
System-luokka; 81

T

Takaisinkutsuttavafunktio; 34, 77
Tavukoodi; 25, 29
TextArea-luokka; 219
TextComponent-luokka; 217
TextField-luokka; 216
This-muuttuja; 74
Thread-luokka; 249
Throw-lause; 82
Tokeni; 29
Toolkit-luokka; 289
Try-catch-lause; 82
Try-finally-lause; 82

U

UNICODE; 38
update-funktio; 157
Update-funktio; 277, 303
URL-osoite; 111

V

Varatut sanat; 37
Vertailuoperaattorit; 49
While-lause; 56
Vierityspalkki; 224
Windows-ohjelmointi; 34
Virus; 25

Y

Ylikirjoittaminen; 74
Ylikuormittaminen; 73